

**Antonis Bikakis
Adrian Giurca (Eds.)**

LNCS 7438

Rules on the Web: Research and Applications

**6th International Symposium, RuleML 2012
Montpellier, France, August 2012
Proceedings**

 **Springer**

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Antonis Bikakis Adrian Giurca (Eds.)

Rules on the Web: Research and Applications

6th International Symposium, RuleML 2012
Montpellier, France, August 27-29, 2012
Proceedings



Springer

Volume Editors

Antonis Bikakis
University College London
Department of Information Studies
Gower Street
London, WC1E 6BT, UK
E-mail: a.bikakis@ucl.ac.uk

Adrian Giurca
Brandenburg University of Technology at Cottbus
Department of Databases and Information Systems
Walther Pauer Str. 2
03046 Cottbus, Germany
E-mail: giurca@tu-cottbus.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32688-2 e-ISBN 978-3-642-32689-9
DOI 10.1007/978-3-642-32689-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012944208

CR Subject Classification (1998): I.2.4, H.3.5, I.2.6, D.2, I.2.11, H.4.1,
F.3.2, D.1.6, J.1

LNCS Sublibrary: SL 2 – Programming and Software Engineering

© Springer-Verlag Berlin Heidelberg 2012

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The International Symposium on Rules, RuleML, has evolved from an annual series of international workshops held since 2002, international conferences in 2005 and 2006, and international symposia since 2007. RuleML 2012, the sixth symposium of this series, brought together researchers and practitioners from industry, academia, and the broader AI community, and presented new research results and applications in the field of rules. It was collocated with the 20th biennial European Conference on Artificial Intelligence (ECAI 2012) in Montpellier, France.

RuleML 2012 was created, inspired, and supported by the RuleML Initiative. RuleML (<http://ruleml.org>) is a non-profit umbrella organization. It consists of several technical groups organized by representatives from academia, industry, and public sectors working on rule technologies and applications. Its aim is to promote the study, research, and application of rules in heterogeneous, distributed environments, such as the Web. RuleML acts as an intermediary between various “specialized” rule vendors, industrial and academic research groups, as well as standardization bodies such as W3C, OMG, OASIS, and ISO. One of its major contributions is the Rule Markup Language, a unifying family of XML-serialized rule languages spanning across all industrially relevant kinds of Web rules.

The technical program of RuleML 2012 included presentations of novel rule-based technologies, such as rule languages, visual languages, mark-up languages, rule engines, formal and operational semantics as well as standardization efforts. It was organized in six main research tracks: Business Rules and Processes, Rule-Based Event Processing and Reaction Rules, Rule-Based Policies and Agents on the Pragmatic Web, Rules and the Semantic Web, Rule Mark-Up Languages and Rule Interchange, and Rule Transformation, Extraction and Learning. These tracks reflect the significant role of rules in several research and application areas, which includes: processing Semantic Web data, enabling the automation of business processes, modeling and reasoning over interactions among agents on the Pragmatic Web, reasoning over actions and events and developing reactive systems, and specifying norms and policies for Web and corporate environments.

Special highlights of this year’s RuleML Symposium included three keynote talks from:

- Robert Kowalski from Imperial College London (with co-author Fariba Sadri) presenting a logic-based framework for reactive systems
- Marie-Laure Mugnier presenting a framework for ontology-based query answering with existential rules
- François Briant from IMB France describing the RIDER (Research for IT Driven EnerGy efficiency) project

The program also included the 6th International Rule Challenge, which was dedicated to practical experiences with rule-based applications, and the RuleML 2012 Doctoral Consortium, which focused on PhD research in the area of rules and mark-up languages.

The contributions in this volume include one paper and one abstract for the first two keynote presentations, two track papers (one by Hans Weigand and Adrian Paschke on the role of rules in the Pragmatic Web, and one by Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian, and Tara Athan on Reaction RuleML, a standardized rule mark-up/serialization language for reaction rules and rule-based event processing), as well as a selection of 14 full papers and eight short papers, including the paper “A Production Rule-Based Framework for Causal and Epistemic Reasoning” by Theodore Patkos, Abdelghani Chibani, Dimitris Plexousakis, and Yacine Amiratm, which was awarded by RuleML with the Best Paper Award, and was invited for presentation at the 26th AAAI Conference (AAAI-12).

We would like to thank all colleagues who submitted papers to RuleML 2012, as well as our Steering Committee and our colleagues from ECAI 2012 for their cooperation and partnership. We would also like to express our gratitude to the Chairs, Program Committee members, and additional reviewers who ensured that this year’s symposium maintained the highest standards of scientific quality. Special thanks go to our keynote speakers, and to our sponsors and partners for their contribution to the success of RuleML 2012. Last, but not least, we would like to thank the development team of the EasyChair conference management system and our publisher, Springer, for their support in the preparation of this volume and the publication of the proceedings.

August 2012

Antonis Bikakis
Adrian Giurca

Organization

General Chairs

Grigoris Antoniou	University of Huddersfield, UK
Guido Governatori	NICTA, Australia

Steering Chairs

Mike Dean	Raytheon BBN Technologies, USA
John Hall	Model Systems, UK
Christian de Sainte Marie	IBM ILOG, France

Program Chairs

Antonis Bikakis	University College London, UK
Adrian Giurca	Brandenburg University of Technology, Germany

Local Chair

Madalina Croitoru	University of Montpellier II, France
-------------------	--------------------------------------

Publicity Chair

Frank Olken	Frank Olken Consulting, USA
-------------	-----------------------------

Metadata and Social Media Chair

Petros Stefaneas	National Technical University of Athens, Greece
------------------	--

Rule Responder Symposium Planner Chair

Chaudhry Usman Ali	University of New Brunswick, Fredericton, Canada
--------------------	---

International Rule Challenge Steering Committee

Patrick Albert	IBM, France
François Briant	IBM CAS, France

International Rule Challenge Chairs

Hassan Aït-Kaci	IBM, Canada
Yuh-Jong Hu	National Chengchi University, Taiwan
Dumitru Roman	SINTEF, Norway

Dotoral Consortium Chairs

Grzegorz J. Nalepa	AGH University of Science and Technology, Poland
Yuh-Jong Hu	National Chengchi University, Taiwan
Monica Palmirani	CIRSFID-University of Bologna, Italy

Track Chairs

Business Rules and Processes

Patrick Albert	IBM, France
----------------	-------------

Rule-Based Event Processing and Reaction Rules

Harold Boley	University of New Brunswick, Canada
--------------	-------------------------------------

Rule-Based Policies and Agents on the Pragmatic Web

Adrian Paschke	Freie Universität Berlin, Germany
Hans Weigand	Tilburg University, The Netherlands

Rules and the Semantic Web

Grzegorz J. Nalepa	AGH University of Science and Technology, Poland
--------------------	---

Rule Mark-Up Languages and Rule Interchange

Nick Bassiliades	Aristotle University of Thessaloniki, Greece
------------------	--

Rule Transformation, Extraction, and Learning

Monica Palmirani	CIRSFID-University of Bologna, Italy
------------------	--------------------------------------

Program Committee

Hassan Aït-Kaci	Martin Atzmueller
Rajendra Akerkar	Costin Badica
Patrick Albert	Ebrahim Bagheri
Darko Anicic	Matteo Baldoni
Alexander Artikis	Nick Bassiliades

Bernhard Bauer
 Yevgen Biletskiy
 Pedro Bizarro
 Luiz Olavo Bonino Da Silva Santos
 Lars Braubach
 Jan Broersen
 Christoph Bussler
 Federico Chesani
 Horatiu Cirstea
 Matteo Cristani
 Claudia D'Amato
 Célia Da Costa Pereira
 Christian De Sainte Marie
 Juergen Dix
 Schahram Dustdar
 Jenny Eriksson Lundström
 Vadim Ermolayev
 Opher Etzion
 Luis Ferreira Pires
 Michael Fink
 Nicoletta Fornara
 Enrico Francesconi
 Fred Freitas
 Aldo Gangemi
 Dragan Gasevic
 Christophe Gravier
 Giancarlo Guizzardi
 Ioannis Hatzilygeroudis
 Stijn Heymans
 Pascal Hitzler
 Yuh-Jong Hu
 Minsu Jang
 Krzysztof Janowicz
 Eric Jui-Yi Kao
 Rainer Knauf
 Paul Krause

Wolfgang Laun
 Domenico Lembo
 Francesca Alessandra Lisi
 Thomas Lukasiewicz
 Michael Maher
 Angelo Montanari
 Chieko Nakabasami
 Grzegorz J. Nalepa
 Frank Olken
 Georgios Paliouras
 Monica Palmirani
 Jose Ignacio Panach Navarrete
 Jeffrey Parsons
 Adrian Paschke
 Axel Polleres
 Fabio Porto
 Alun Preece
 Dave Reynolds
 Pierangela Samarati
 Giovanni Sartor
 Guy Sharon
 Davide Sottara
 Giorgos Stamou
 Kostas Stathis
 Giorgos Stoilos
 Nenad Stojanovic
 Umberto Straccia
 Terrance Swift
 Leon Van Der Torre
 Jan Vanthienen
 Wamberto Vasconcelos
 Carlos Viegas Damásio
 George Vouros
 Renata Wassermann
 Ching-Long Yeh

External Reviewers

Amina Chniti
 Minh Dao-Tran
 Benjamin Jailly
 Stasinou Konstantopoulos

Jens Lehmann
 Abhijeet Mohapatra
 Kunal Sengupta
 Anastasios Skarlatidis

RuleML 2012 Sponsors and Partners

Gold Sponsors



ALMA MATER STUDIORUM
UNIVERSITA DI BOLOGNA

Bronze Sponsors



Model Systems

Partner Organizations

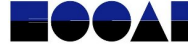


Table of Contents

Keynote Talks

A Logic-Based Framework for Reactive Systems	1
<i>Robert Kowalski and Fariba Sadri</i>	
Ontology-Based Query Answering with Existential Rules	16
<i>Marie-Laure Mugnier</i>	

Business Rules and Processes

A Model Driven Reverse Engineering Framework for Extracting Business Rules Out of a Java Application	17
<i>Valerio Cosentino, Jordi Cabot, Patrick Albert, Philippe Bauquel, and Jacques Perronnet</i>	
Business Process Data Compliance	32
<i>Mustafa Hashmi, Guido Governatori, and Moe Thandar Wynn</i>	
Formalizing Both Refraction-Based and Sequential Executions of Production Rule Programs	47
<i>Bruno Berstel-Da Silva</i>	
Bringing OWL Ontologies to the Business Rules Users	62
<i>Adil El Ghali, Amina Chniti, and Hugues Citeau</i>	
From Regulatory Texts to BRMS: How to Guide the Acquisition of Business Rules?	77
<i>Abdooulaye Guissé, François Lévy, and Adeline Nazarenko</i>	
A Rule Based Approach for Business Rule Generation from Business Process Models	92
<i>Saleem Malik and Imran Sarwar Bajwa</i>	

Rule-Based Event Processing and Reaction Rules

Reaction RuleML 1.0: Standardized Semantic Reaction Rules	100
<i>Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian, and Tara Athan</i>	
A Production Rule-Based Framework for Causal and Epistemic Reasoning	120
<i>Theodore Patkos, Abdelghani Chibani, Dimitris Plexousakis, and Yacine Amirat</i>	

On the Algebraic Semantics of Reactive Rules	136
<i>Katerina Ksytra, Nikolaos Triantafyllou, and Petros Stefanos</i>	
A Rule-Based Calculus and Processing of Complex Events	151
<i>Stefano Bragaglia, Federico Chesani, Paola Mello, and Davide Sottara</i>	
Complex Reactivity with Preferences in Rule-Based Agents	167
<i>Stefania Costantini and Giovanni De Gasperis</i>	

Rule-Based Policies and Agents on the Pragmatic Web

The Pragmatic Web: Putting Rules in Context	182
<i>Hans Weigand and Adrian Paschke</i>	
HARM: A Hybrid Rule-Based Agent Reputation Model Based on Temporal Defeasible Logic	193
<i>Kalliopi Kravari and Nick Bassiliades</i>	
Whispering Interactions to the End User Using Rules.....	208
<i>Benjamin Jailly, Christophe Gravier, Julien Subercaze, Marius Preda, and Jacques Fayolle</i>	
Personalizing Location Information through Rule-Based Policies	215
<i>Iosif Viktoratos, Athanasios Tsadiras, and Nick Bassiliades</i>	

Rules and the Semantic Web

Using <i>SOIQ(D)</i> to Formalize Semantics within a Semantic Decision Table.....	224
<i>Yan Tang Demey and Trung-Kien Tran</i>	
Imposing Restrictions over Temporal Properties in OWL: A Rule-Based Approach	240
<i>Sotiris Batsakis and Euripides G.M. Petrakis</i>	
OWL RL in Logic Programming: Querying, Reasoning and Inconsistency Explanations	248
<i>Jesús M. Almendros-Jiménez</i>	
Using Data-to-Knowledge Exchange for Transforming Relational Databases to Knowledge Bases	256
<i>Tadeusz Pankowski</i>	

Rule Markup Languages and Rule Interchange

PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners	264
<i>Gen Zou, Reuben Peter-Paul, Harold Boley, and Alexandre Riazanov</i>	
PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes	280
<i>Mohammad Sadnan Al Manir, Alexandre Riazanov, Harold Boley, and Christopher J.O. Baker</i>	
Syntax Reuse: XSLT as a Metalanguage for Knowledge Representation Languages	289
<i>Tara Athan</i>	

Rule Transformation, Extraction and Learning

An Approach to Parallel Class Expression Learning	302
<i>An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland</i>	
Rule-Based High-Level Situation Recognition from Incomplete Tracking Data	317
<i>David Münch, Joris IJsselmuiden, Ann-Kristin Grosselfinger, Michael Arens, and Rainer Stiefelhagen</i>	
Author Index	325

A Logic-Based Framework for Reactive Systems

Robert Kowalski and Fariba Sadri

Department of Computing
Imperial College London
{rak, fs}@doc.ic.ac.uk

Abstract. We sketch a logic-based framework in which computation consists of performing actions to generate a sequence of states, with the purpose of making a set of reactive rules in the logical form *antecedents* \rightarrow *consequents* all true. The *antecedents* of the rules are conjunctions of past or present conditions and events, and the *consequents* of the rules are disjunctions of conjunctions of future conditions and actions. The *antecedents* can be viewed as complex/composite events, and the *consequents* as complex/composite/macro actions or processes.

States are represented by sets of atomic sentences, and can be viewed as global variables, relational databases, Herbrand models, or mental representations of the real world. Events, including actions, transform one state into another. The operational semantics maintains only a single, destructively updated current state, whereas the model-theoretic semantics treats the entire sequence of states, events and actions as a single model. The model-theoretic semantics can be viewed as the problem of generating a model that makes all the reactive rules true.

Keywords: reactive systems, model generation, LPS, KELPS, complex events, complex actions.

1 Introduction

Reactive rules in one form or another play an important role in many different areas of Computing. They are explicit in the form of condition-action rules in production systems, event-condition-action rules in active databases, and plans in BDI agents. Moreover, they are implicit in many other areas of Computing.

David Harel [7], for example, identifies reactive systems as one of the main kinds of computational system, characterizing them as having the general form “when event α occurs in state A , if condition C is true at the time, the system transfers to state B ”. Harel [8] notes that StateCharts, a graphical language for specifying reactive systems, is “the heart of the UML - what many people refer to as its driving behavioral kernel”.

Reactive systems can be regarded as an extension of transition systems. As Wolfgang Reisig [22] puts it, an initialized, deterministic transition system is “a triple $C = (Q, I, F)$ where Q is a set (its elements are denoted as states), $I \subseteq Q$ (the initial states), and $F : Q \rightarrow Q$ (the next-state function)”. Transition systems can be extended to reactive systems in which the transition from one state to the next is “not conducted by the program, but by the outside world”.

But even in their simpler “initialized, deterministic” form, transition systems have been proposed as a general model of Computing. Reisig points out that, in the first volume of *The Art of Computer Programming* [12], Donald Knuth suggests their use as a general semantics for algorithms.

In this paper, we propose a simplified kernel, KELPS, of the Logic-based agent and Production System language LPS [15, 16, 17]. Whereas LPS combines reactive rules and logic programs, KELPS consists of reactive rules alone. The distinguishing features of LPS and KELPS are that:

- They interpret reactive rules of the form *antecedents* \rightarrow *consequents* as universally quantified sentences in first-order logic (FOL).
- They have an operational semantics in which actions and other events destructively update a current state.
- They have a model-theoretic semantics in which actions are performed to make the reactive rules all true in a single classical model associated with the sequence of states and events.

The aim of this paper is to highlight the significance of reactive rules in LPS, and to take advantage of the simplicity of KELPS to introduce some extensions to LPS. The main extensions are the addition of complex events in the antecedents of reactive rules, the generalization of the sequential ordering of conditions and events to partial ordering, and the representation of conditions by arbitrary formulas of FOL.

The paper is structured as follows: Section 2 introduces KELPS by means of an example. Section 3 sketches the background of LPS and KELPS. Sections 4-5 describe the syntax, model-theoretic and operational semantics of KELPS; and section 6 discusses soundness and incompleteness. Section 7 describes the extension of KELPS by logic programs; and section 8 sketches an extension to the multi-agent case. Sections 9 and 10 conclude.

2 Example

We use a variant of an example in [9]. In this variant, a reactive agent monitors a building for any outbreaks of fire. The agent receives inputs from a fire alarm pre-sensor and a smoke detector. If the agent detects a possible fire, then it activates local fire suppression devices and in the case that there is no longer a possible fire, calls for a security guard to inspect the area. Alternatively, it calls the fire department. This behaviour can be represented in KELPS by means of a reactive rule such as:

*if pre-sensor detects possible fire in area A at time T_1
 and smoke detector detects smoke in area A at time T_2
 and $|T_1 - T_2| \leq 60 \text{ sec}$ and $\max(T_1, T_2, T)$
 then [[activate local fire suppression in area A at time T_3 and $T < T_3 \leq T + 10 \text{ sec}$
 and not fire in area A at time T_4 and $T_3 < T_4 \leq T_3 + 30 \text{ sec}$
 and send security guard to area A at time T_5 and $T_4 < T_5 \leq T_4 + 10 \text{ sec}$]
 or [call fire department to area A at time T_3' and $T < T_3' \leq T + 60 \text{ sec}$]*

Here A , T_1 , T_2 and T are implicitly universally quantified with scope the entire rule, but T_3 , T_3' , T_4 and T_5 are implicitly existentially quantified with scope the consequents of the rule.

The notation employed in this example is an informal version of the internal syntax of KELPS with an explicit representation of time. It is compatible with a variety of external notations that hide the representation of time. We employ this internal syntax throughout the paper, as it clarifies the operational and model-theoretic semantics.

Notice that the antecedents of the rule represent a complex event and the consequents represent a complex action consisting of two alternative plans. The first plan consists of two actions and a condition, and the second consists of an action. Both plans include temporal constraints. In practice, the first plan might be preferred and tried before the second. If any part of the first plan fails, then the second plan can be tried. Moreover, even if some of the first plan fails, it can be retried as long as the temporal constraints can be satisfied. If both plans fail and cannot be retried, then the reactive rule cannot be made true. This can be avoided by adding additional alternative plans to the consequents of the rule.

3 LPS and KELPS in Context

LPS is a direct descendant of ALP agents [14], which descend in turn from abductive logic programming (ALP). ALP extends logic programs with undefined (open or abducible) predicates, which are constrained by integrity constraints [11]. ALP agents embed ALP in an agent cycle, in which logic programs serve as the agent's *beliefs*, and integrity constraints serve as the agent's *goals*.

In ALP agents, the world is represented, as in the situation calculus [20], event calculus [19] and other knowledge representation schemes in AI, by atomic predicates with time or state parameters, called *fluents*. Updates of the world are described by an *event theory* that specifies the fluents that are initiated and terminated by events. *Frame axioms* of one form or another express the property that if a fluent holds in a state, then it continues to hold in future states unless and until it is terminated by an event.

Frame axioms used to reason about states of affairs exact a heavy computational penalty, which is prohibitively large for even moderately sized knowledge bases. As a consequence, frame axioms are rarely used in practical applications.

ALP agents use the event calculus with its frame axiom to reason about fluents. However, ALP agents also have the option of observing fluents, instead of reasoning to derive them. This is one step towards eliminating frame axioms entirely. The next and final step, which historically resulted in LPS, came from trying to simulate production systems by means of ALP agents.

This final step was made difficult by the overabundance of semantics for ALP. The operational semantics of ALP agents, in particular, is based on the IFF proof procedure [6], which is complete for the Kunen three-valued semantics. The attractiveness of this completeness result was an obstacle to identifying a semantics that is better suited for production systems. Eventually it became apparent that the necessary semantics is one of the simplest possible, namely the minimal model semantics of Horn clauses [3].

In LPS, an agent's beliefs are represented by logic programs, and the agent's goals are represented by reactive rules, which are a special case of integrity constraints in ALP. The purpose of the agent's actions is to extend its beliefs about the world, so that its reactive rules are all true in the minimal model of the extended beliefs. The minimal model is unique if the beliefs are represented by Horn clauses [3]. If the beliefs are represented by locally stratified logic programs [21], then the extended beliefs have a unique intended, minimal model, called the perfect model.

In retrospect, KELPS is probably closest to the modal agent language MetateM. In MetateM a program consists of sentences in logical form:

‘past and present formula’ **implies** ‘present or future formula’ [4]

and computation consists of generating a model in which all such sentences are true. These sentences are represented in a modal temporal logic, with truth values defined relative to a possible world embedded in a collection of possible worlds connected by a temporal accessibility relation. In contrast, in KELPS, similar sentences are expressed in classical FOL with an explicit representation of time, with truth values defined relative to a single classical model, in the standard Tarskian manner. Moreover, whereas MetateM uses frame axioms to reason about possible worlds, KELPS uses destructive updates to transform one state into another.

In other respects, full LPS is closer to Transaction Logic (\mathcal{TR} Logic) [1], which also maintains only the current state, using destructive updates without frame axioms. Moreover, like LPS and MetateM, \mathcal{TR} Logic also gives a model-theoretic semantics to complex actions (and transactions), by interpreting them as sentences in logical form and by generating a model in which these sentences are true.

However, in \mathcal{TR} Logic, transactions are expressed in a non-standard logic with logical connectives representing temporal sequence. The model in \mathcal{TR} Logic is a collection of possible worlds, as in modal logic. But truth values are defined relative to paths between possible worlds.

In LPS, in contrast with both MetateM and \mathcal{TR} Logic, truth values are defined relative to a single, classical model, which contains the entire sequence of states and state-transforming events. Possible worlds in the semantics of MetateM and \mathcal{TR} Logic become sub-models of this single model in LPS, and the accessibility relation becomes a relation expressed by means of event descriptions and temporal constraints represented explicitly in the language. This simplifies the semantics and increases the expressive power of the language by making events and times first class objects.

4 KELPS and Its Model-Theoretic Semantics

Viewed in agent-oriented terms, *reactive rules* in KELPS are *maintenance goals* expressed in logical form. Their truth values are determined by the agent's *beliefs*, which include an initialised sequence of states $S_0, S_1, \dots, S_i, \dots$ and an associated sequence of state-transforming sets of events e_1, \dots, e_i, \dots .

States S_i in KELPS are sets of atomic sentences, also called *facts* or *fluents*. They can be understood as representing sets of global variables, relational databases, Herbrand models, or mental representations of the real world. Events e_i are also sets of atomic sentences, and represent either external events or the agent's own actions.

4.1 The Syntax of Reactive Rules

In the model-theoretic semantics, facts, events and reactive rules are represented with time-stamps. However, in the operational semantics, facts are represented without time-stamps, so that facts that are not affected by an event simply persist from one state to the next without needing to reason with frame axioms that they persist.

Correspondingly, reactive rules can also be represented either in an internal syntax with time parameters, or in an external syntax without explicit time. The internal syntax underpins the model-theoretic semantics, and is consequently more basic. It also facilitates more flexible and more powerful ways of representing and reasoning with temporal constraints. For these reasons we focus on the internal syntax in this paper.

At the expense of restricting its expressive power, the internal syntax is compatible with a variety of external notations. In [16, 17], we specified an external syntax in which temporal ordering is indicated by the order in which formulas are written and by special logical connectives. However, the internal syntax is also compatible with other external notations, such as the syntax of \mathcal{TR} Logic, in which $P \otimes Q$ means “do P and then do Q ”. In LPS/KELPS this translates into $P(T_1) \wedge Q(T_2) \wedge T_1 < T_2$.

The internal syntax is also compatible with a modal external syntax. For example, $P \wedge \diamond Q$ can also be translated into $P(T_1) \wedge Q(T_2) \wedge T_1 < T_2$. Graphical notations for partial ordering are also possible.

Here we define the *internal syntax* of the reactive rules. A *reactive rule* (or more simply, a *rule*) is a universally quantified conditional sentence of the form¹:

$$\forall X [antecedents(X) \rightarrow \exists Y consequents(X, Y)]$$

X is the set of all time variables and any other unbound variables that occur in *antecedents*, and Y is the set of all time variables and any other unbound variables that occur only in the *consequents*. This convention allows us to omit the explicit representation of these two quantifiers and write rules in the simpler form:

$$antecedents(X) \rightarrow consequents(X, Y)$$

To ensure that the actions and conditions in *consequents*(X, Y) occur after the events and conditions in *antecedents*(X), the time variables in Y should be constrained directly or indirectly in *consequents* to be later than the time variables in X .

Antecedents of rules can be viewed as complex (or composite) events, and *consequents* can be viewed as complex actions (or processes). The *antecedents* are a conjunction, each conjunct of which is either a condition, an event atom or a temporal constraint, where:

¹ This notation means that X includes all the variables that occur in the *antecedents*(X). But it does not mean that *consequents*(X, Y) contains all of the variables in X .

- A *condition* is an FOL formula including only a single time parameter in the vocabulary of the state predicates, possibly including auxiliary state-independent predicates that do not change with time.²
- An *event atom* is an atomic formula with a single time parameter representing the occurrence of an event. Similarly an *action atom* is an event atom in which the event is an action.
- A temporal constraint is an inequality $time_1 < time_2$ or $time_1 \leq time_2$, where $time_1$ and $time_2$ represent time points, one of which is a variable, and the other of which is a variable or constant.³

The *consequents* of a rule are a disjunction, each disjunct of which is a conjunction, each conjunct of which is either a condition, an action atom or a temporal constraint. Temporal constraints in the *antecedents* should involve only time variables occurring elsewhere in the *antecedents*, and temporal constraints in the *consequents* should involve only time variables occurring elsewhere in the rule.⁴

The various restrictions on the syntax described above simplify the operational semantics presented later. For example, these restrictions limit complex events to partially ordered sequences of conditions and simpler events. In particular, they do not allow complex events that include a constraint that no event of a certain kind occurs within a certain period of time.⁵ Moreover, they do not allow external events in the consequents of rules.

Both of these restrictions can be removed at the expense of complicating the operational semantics. In fact, as we will see in the next section, the syntax of reactive rules can be generalized to include arbitrary sentences of FOL.

4.2 The Model-Theoretic Semantics

Viewed in general terms, the task in KELPS is to generate a Herbrand model that makes a set of sentences in FOL all true. The core of the model is determined by a sequence of states $S_0, S_1, \dots, S_i, \dots$ and state-transforming sets of events e_1, \dots, e_i, \dots . These sequences are combined into a single model by time-stamping facts and events.

The model-theoretic semantics is compatible with a variety of different notions of time and temporal ordering. However, for simplicity in this paper, we assume that time is discrete, and that events e_i are instantaneous, and are stamped with the time t_i of their occurrence, written either as $e_i(t_i)$, *happens*(e_i, t_i) or simply e_i^* .

We also assume that the time t_i at the beginning of a state S_i “lasts” until the time t_{i+1} at the end of state S_i . So in effect $t = t_i$ for all $t_i \leq t < t_{i+1}$. The time-stamping of a

² For example, *Aux* might contain atomic sentences defining a predicate *area A is connected to area B*, which might be useful in a refinement of the reactive rule in section 2.

³ We also need to allow arithmetic expressions such as $t + n$. This can be done by replacing such expressions by variables, and by including auxiliary conditions that perform the necessary arithmetic. E.g. replace $t + n$ by a variable T , and add a condition *plus*(t, n, T).

⁴ To simplify both the model-theoretic and operational semantics we need to impose a *range restriction* on conditions, preventing such rules as $\neg p(X, T_1) \rightarrow q(X, Y, T_2) \wedge T_1 < T_2$, in which the variable X is unrestricted.

⁵ E.g. the complex event *stock goes up more than 5% at time T_1 and stock goes up more than 5% at a later time T_2 and stock does not go down more than 2% between T_1 and T_2 .*

fact p that is true in a state S_i is written either as $p(t_i)$ or as $holds(p, t_i)$. The resulting state S_i in which all its facts have been time-stamped is written simply as S_i^* .

Because the conditions and events in the *antecedents* and *consequents* of reactive rules are temporally ordered by inequality constraints between time points, the model-theoretic semantics needs a specification of the inequality relation. In the operational semantics, the inequality relation can be computed by any means that respects this specification, including the use of temporal constraint processing. However, in the model theory, the specification of the temporal ordering is represented extensionally by an infinite set of atomic sentences $t \leq t'$ and $t < t'$ for time points.

This extensional representation of the inequality relations is included in a set Aux , which also includes extensional representations of any other auxiliary predicates, such as absolute value $||$ and max in the example of section 2. In LPS, these auxiliary predicates are defined by logic programs.

With this time-stamping of facts and events, and this extensional representation of the inequality relation and of any other auxiliary predicates, the sequence of states and events can be combined into a single Herbrand model:

$$M = Aux \cup S_0^* \cup e_1^* \cup S_1^* \cup e_2^* \cup \dots \cup e_i^* \cup S_i^* \cup e_{i+1}^* \dots$$

In general, a *Herbrand model*, which is a set of atomic sentences, has a dual interpretation. As a set of sentences, it is a purely syntactic object. But, as a specification of the set of all atomic sentences that are true in the *Herbrand universe* (i.e. the set of all variable-free terms that can be constructed from the vocabulary of the language), it is a model-theoretic structure. These two interpretations are closely related: A Herbrand model, viewed as a model-theoretic structure, is the *unique minimal model* of itself, viewed syntactically as a set of sentences. As we will see in section 7, this minimal model relationship is the key to the semantics of the more general case, in which the set Aux is generalized to a logic program.

The definition of truth in a Herbrand model for arbitrary sentences of FOL follows the classical Tarskian definition. This includes the definition of truth for reactive rules: A rule of the form $\forall X [antecedents(X) \rightarrow \exists Y consequents(X, Y)]$ is true in M if and only if, for every ground instance x over the Herbrand universe of the variables X , if $antecedents(x)$ is true in M , then there exists a ground instance y over the Herbrand universe of the variables Y such that $consequents(x, y)$ is true in M .

Notice that, in theory, because truth is defined for arbitrary sentences of FOL, the set of reactive rules could be replaced by any set of FOL sentences. The syntactic restrictions on reactive rules have been imposed primarily to simplify the operational semantics so that it operates with only a single current state.⁶

Given a set of reactive rules R , an initial state S_0 and a sequence of sets of external events ex_1, \dots, ex_i, \dots , the *task* in KELPS is to generate an associated sequence of sets of actions a_1, \dots, a_i, \dots such that all of the reactive rules in R are true in the resulting

⁶ For example, in the operational semantics, only the last set of events e_i is accessible during the i -th cycle. This restriction can be liberalised to include a window of previous events, with the benefit that more complex events can be catered for without too much loss of efficiency.

Herbrand model $M = Aux \cup S_0^* \cup e_1^* \cup S_1^* \cup e_2^* \cup \dots \cup e_i^* \cup S_i^* \cup e_{i+1}^* \dots$. Here $e_i = ex_i \cup a_i$ and S_{i+1} is obtained from S_i by adding any fluents initiated by the events in e_{i+1} and by deleting any fluents terminated by the events in e_{i+1} .

5 KELPS Operational Semantics

The model-theoretic semantics of KELPS is compatible with many different operational semantics. The operational semantics sketched in this section uses the internal syntax, and employs an observe-think-decide-act agent cycle. It is relatively abstract, and is itself compatible with many different implementations.

We assume that the i -th cycle starts at time t_i , when the events e_i transform the state S_{i-1} into the state S_i , and that it ends at time $t_{i+1} = t_i + \varepsilon_i$, where ε_i is the duration of the cycle. We assume that this duration ε_i is short enough that actions can be executed in a timely manner, and long enough that all the relevant conditions can be evaluated in the current state within a single cycle. In addition, we assume that the state S_i remains unchanged throughout the period from t_i to t_{i+1} i.e. $t = t_i$ for all $t_i \leq t < t_{i+1}$. This is so that the truth values of conditions evaluated in S_i are determined relative to a single time point t_i , but their evaluation can take place during the interval from t_i to t_{i+1} . Any events are observed and assimilated only at the beginning of cycles.

In addition to maintaining the current state S_i , the operational semantics maintains a *goal state* G_i , which is a set of *achievement goals*, each of which can be regarded as a set of alternative conditional plans of actions for the future. Logically each goal state is a conjunction, and each conjunct is a disjunction of existentially quantified conjunctions of temporally constrained conditions and actions. These achievement goals are typically derived from the *consequences* of reactive rules (maintenance goals). However, they can also be given separately in the initial goal state G_0 .

Operationally each achievement goal in the goal state can be regarded as a separate thread. In particular different threads, and different alternatives within the same thread, do not share any variables.⁷

To deal with complex events in the *antecedents* of reactive rules, the operational semantics also maintains a current set of reactive rules R_i . A new rule is added to R_i when an instance of a condition or event in the *antecedents* of a rule is true in the current state. The new rule is an instance of the rest of the original rule that needs to be maintained in the future.

To simplify the operational semantics, and to avoid over-constraining times, we exclude conditions and events of the form $p(t)$ in reactive rules, where t is a time constant. Instead, we write $p(T) \wedge t \leq T \leq t + \delta$ for some suitably small δ .

With these simplifying assumptions, the operational semantics begins with an initial state S_0 , goal state G_0 and set of reactive rules R_0 . The i -th cycle, starting with state S_{i-1} , goal state G_i , rules R_i and events e_i at time t_i consists of the following steps:

Step 0. Observe. State S_{i-1} is transformed into state S_i by adding and deleting all the facts associated with all the events in e_i .

⁷ Note that $\exists Y[p(Y) \vee q(Y)]$ is equivalent to $\exists Y p(Y) \vee \exists Z q(Z)$.

Step 1. React. For every reactive rule in R_i of the form:

$$\text{antecedent}(X) \wedge \text{other-antecedents}(X) \rightarrow \text{consequents}(X, Y)$$

where $\text{antecedent}(X)$ is a conjunction of conditions and event atoms such that there are no conditions or event atoms in $\text{other-antecedents}(X)$ constrained to be earlier or at the same time as those in $\text{antecedent}(X)$,⁸ the operational semantics finds all ground instances $\text{antecedent}(x)$ of $\text{antecedent}(X)$ that are true in $e_i^* \cup S_i^*$ viewed as a Herbrand model.⁹ For each such instance, it generates the corresponding “resolvent”:

$$\text{other-antecedents}(x) \rightarrow \text{consequents}(x, Y)$$

instantiating all the time variables $Times$ that occur in $\text{antecedent}(X)$ to the current time t_i , simplifying any temporal constraints, and adding the simplified resolvent as a new reactive rule to R_i . Simplification consists in deleting any inequalities that are true in Aux . R_{i+1} is the resulting expanded set of reactive rules.

If after simplification, $\text{other-antecedents}(x)$ is an empty conjunction (equivalent to true), then the goal state G_i is updated by adding the simplified $\text{consequents}(x, Y)$ as a new achievement goal, starting a new thread.

Step 2. Solve Goals

Step 2.1. Decide. The operational semantics deletes from G_i any disjuncts that are now too late, because they contain conditions whose times are constrained to be earlier than t_i or actions whose times are constrained to be earlier or identical to t_i . If, as a result, all the disjuncts in a thread are deleted, then the thread is false, and the operational semantics terminates in failure.

Otherwise, the operational semantics chooses a set D of disjuncts for evaluation/execution from one or more threads in G_i . Choosing disjuncts from different threads amounts to solving several goals concurrently. Choosing disjuncts from the same thread explores different ways of solving the same goal simultaneously.

Step 2.2. Think. For every disjunct in D , the operational semantics chooses a form¹⁰:

$$\text{conditions}(Y) \wedge \text{other-consequents}(Y)$$

where $\text{conditions}(Y)$ is a conjunction of conditions such that there are no conditions or events in $\text{other-consequents}(Y)$ that are constrained to be earlier or at the same time as those in $\text{conditions}(Y)$.

The chosen conditions may be empty (equivalent to true), in which case the operational semantics continues with step 3. Otherwise, it queries the current state S_i to

⁸ This phrasing allows for the possibility that the antecedents of the rule are partially ordered. In such cases, it may be possible to parse a single rule into this form in different ways.

⁹ Note that x instantiates only those variables in X that are in $\text{antecedent}(X)$. Any variables in X that are not in $\text{antecedent}(X)$ remain as variables in x .

¹⁰ Because all of the universally quantified variables have been instantiated in step 1 to variable-free terms, neither they nor their instances are displayed in this notation.

determine *some* instance $conditions(y)$ of $conditions(Y)$ that is true in S_i^* . It then generates the corresponding “resolvent”:

$$other-consequents(y)$$

instantiating the time variables $Times'$ in $conditions(Y)$ to the current time t_i , simplifying any temporal constraints, and updating both D and G_i by adding the simplified resolvent to the thread in D and G_i containing $conditions(Y) \wedge other-consequents(Y)$.¹¹

Step 3. Act. For every disjunct in D the operational semantics chooses a form:

$$actions(Z) \wedge further-consequents(Z)$$

where $actions(Z)$ is a conjunction of action atoms such that there are no conditions or actions in $further-consequents(Y)$ that are constrained to be earlier or at the same time as those in $actions(Z)$.

The chosen $actions$ may be empty (equivalent to true), in which case the operational semantics continues with step 4. Otherwise, it attempts to execute $actions(Z)$. For each such disjunct, if one of the actions in $actions(Z)$ fails, then the conjunction of actions fails, and therefore all the actions in $actions(Z)$ fail. For all the actions that succeed, it generates the corresponding “resolvent”¹²:

$$further-consequents(z)$$

instantiating the time variables $Times''$ that occur in $actions(Z)$ to the new current time t_{i+1} , simplifying any temporal constraints in $further-consequents(z)$, and updating G_i by adding the simplified resolvent to the thread containing $actions(Z) \wedge further-consequents(Z)$.

If a new disjunct is empty (equivalent to true), the operational semantics deletes the thread containing the disjunct (because the thread is then also equivalent to true).

Step 4. The cycle ends. The current goal state G_i becomes the next goal state G_{i+1} , and any successfully executed actions are added to the set of events e_{i+1} .

6 Soundness and Incompleteness

The soundness of the operational semantics of KELPS can be shown by means of a similar argument to the argument for LPS [17]. The incompleteness of KELPS can also be shown by means of similar counterexamples. In particular, the operational semantics of neither LPS nor KELPS can:

¹¹ Notice that $conditions(Y) \wedge other-consequents(Y)$ is retained in G_i , because $conditions(Y)$ can be queried again in the future, as long as no time in $Times'$ becomes constrained to be in the past. Moreover it may be possible to try the same disjunct again with a choice of different $conditions(Y)$. Similarly, $actions(Z) \wedge further-consequents(Z)$ is retained in G_i in step 2.3.

¹² The instance z of the variables in Z can be regarded as a kind of a feedback from the environment.

1. preventively make a reactive rule true by making its *antecedents* false, or
2. proactively make a reactive rule true by making its *consequents* true before its *antecedents* become true.

These two kinds of incompleteness are tolerable for a simple-minded agent, which can compensate for them by the use of explicit, additional reactive rules, to cater independently for these cases. But they are undesirable for a genuinely intelligent agent.

Examples of these two kinds of incompleteness include:

1. $attacks(X, you, T_1) \wedge \neg prepared\text{-for}\text{-}attack(you, T_1)$
 $\rightarrow surrender(you, T_2) \wedge T_1 < T_2 \leq T_1 + \delta$
 KELPS cannot make the rule true by performing actions to make *prepared-for-attack(you, T)* true and so $\neg prepared\text{-for}\text{-}attack(you, T)$ false.
2. $enter\text{-}bus(T_1) \rightarrow have\text{-}ticket(T_2) \wedge T_1 < T_2 \leq T_1 + \varepsilon$
 KELPS cannot make the rule true by performing actions to make *have-ticket(T₂)* true before *enter-bus(T₁)*.

7 KELPS + LP

KELPS makes the simplifying assumption that auxiliary predicates are defined by possibly infinitely many facts *Aux*. This has the advantage that it highlights the primary role of reactive rules. But it ignores the need to define auxiliary predicates in a structured, reusable and computationally feasible manner. This need can be filled very simply by logic programs, whose minimal model and perfect model semantics are a natural extension of the model-theoretic semantics of KELPS.

In addition to their use for defining auxiliary predicates, logic programs can be used to define intensional predicates, like view definitions in relational databases. They can also be used to name complex events and actions and to define them both recursively and in terms of other events and actions.

For example, whenever there is an emergency of any kind, we may want to isolate and monitor the area. This can be represented by the reactive rule:

$$emergency(Area, T_0) \rightarrow isolate(Area, T_1, T_2) \wedge T_0 < T_1 < T_2$$

and logic program:

$$\begin{aligned} emergency(Area, T) &\leftarrow fire(Area, T) \vee flood(Area, T) \vee noxious\text{-}fumes(Area, T) \\ isolate(Area, T_1, T_4) &\leftarrow close\text{-}windows(Area, T_1, T_2) \wedge close\text{-}doors(Area, T_2, T_3) \wedge \\ &lock\text{-}doors(Area, T_3, T_4) \wedge T_1 < T_2 < T_3 < T_4 \wedge focus\text{-}camera(Area, T_5) \wedge T_1 < T_5 \leq T_4 \end{aligned}$$

Consider the simplest case, in which the auxiliary predicates are defined by a set of Horn clauses *LP*. This means that

$$M = Aux \cup S_0^* \cup e_1^* \cup S_1^* \cup e_2^* \cup \dots \cup e_i^* \cup S_i^* \cup e_{i+1}^* \dots$$

is the unique minimal model of the set of Horn clauses [3]:

$$LP \cup S_0^* \cup e_1^* \cup S_1^* \cup e_2^* \cup \dots \cup e_i^* \cup S_i^* \cup e_{i+1}^* \dots$$

The uniqueness of the intended model continues to hold if Horn clause are generalised to locally stratified logic programs [21]. It also continues to hold if negative literals in logic programs are appropriately generalised to formulas of FOL.

As in the case of KELPS, different operational semantics are possible for KELPS + LP. In the case of macro-actions in the consequents of reactive rules, the natural execution method is backward reasoning. However, in the case of macro-events in the antecedents of reactive rules, the natural execution method is a form of forward reasoning triggered by the observation of events. This kind of forward reasoning is similar to integrity checking methods for deductive databases and to the Rete algorithm [5] for production systems. It can be implemented, for example, by means of resolution in the connection graph proof procedure [13].

Earlier papers about LPS [16, 17] contain planning clauses and use intentional predicate definitions to generate future states. The focus in this paper is on defining a version of LPS that is closer to a practical programming language, rather than to a problem description and problem-solving language.

8 MALPS – Multi-Agent LPS

In LPS/KELPS, an agent interacts with a global state, observing and performing updates. This global state can also serve as a communication and coordination medium for a community of agents, like the blackboard model in AI [10], tuple spaces in Linda [2], and conventional, multi-user database management systems.

MALPS (Multi-Agent LPS) [18] exploits this potential of the global state to serve as a coordination medium. Different agents can have different goals, represented by different reactive rules, different beliefs, represented by different logic programs, and different capabilities, represented by different atomic actions.

The extension of LPS/KELPS to MALPS requires no change to the model-theoretic semantics: An action performed by one agent becomes an external event for other agents. Whereas in LPS/KELPS, only a single agent tries to make its own goals true, in MALPS all the agents try to make their goals true.

The use of a global state as a communication and coordination medium contrasts with the use of message-passing in many other computing paradigms, in which the use of global states and global variables is considered an unsafe practice with a problematic semantics. In KELPS, global states not only provide the basis for its operational semantics, but they also underpin its model-theoretic semantics.

The simplest way to implement MALPS is to synchronise updates, so that all of the agent cycles start and end at the same time. This is similar to the simplifying assumption in KELPS that time stands still for the duration of a cycle. This simplifying assumption ensures that, if the only external events are the actions of other agents, then if an agent believes that a condition is true in a given state, then it really is true because no other agent can update the state during the cycle.

9 Conclusions

We have presented a reactive kernel KELPS of LPS that deals with complex events, generalized FOL conditions and temporal constraints.

As we have already mentioned, KELPS is similar to MetateM, with their common focus on generating a model to make a collection of reactive rules true. However, MetateM differs from KELPS by employing modal logic syntax with a possible world semantics, and by using frame axioms to implement change of state.

LPS is also similar to \mathcal{TR} Logic, with their common use of a destructively updated state, and their common view of complex actions as sequences of conditions that are solved by queries and atomic actions that are performed by updates. Indeed, the fact that conditions in LPS can be FOL formulas is largely inspired by \mathcal{TR} Logic.

\mathcal{TR} Logic employs a non-modal syntax, but with a possible world semantics, in which truth values are determined relative to paths in a collection of possible worlds. This semantics is natural for transactions. But it seems cumbersome for reactive rules, whose truth value then depends on whether, for every path over which the *antecedents* are true, there exists a later path over which the *consequents* are true.

MetateM and Transaction Logic are the two computational frameworks that are closest to LPS/KELPS, but they are only the tip of an iceberg. Numerous other attempts have been made to develop a model-theoretic semantics for state transition systems. As far as we can tell, none of them view computation as generating a classical FOL model of a program.

The model-theoretic semantics of FOL, upon which the semantics of LPS is based, is the simplest model-theoretic semantics possible, because all other model-theoretic semantics, including the possible worlds semantics, reduce to it. The use of classical FOL semantics is possible for LPS, because the internal syntax of LPS includes explicit representations of times and events, making it possible to generate single models that include the entire sequence of states and events. The representation of time in the internal syntax can be partially hidden in an external syntax, but can be brought to the surface when necessary, for example to refer to temporal constraints on times and durations. This ability to represent and reason about temporal constraints is essential for practical applications, including those that involve real time and scheduling.

Those frameworks, other than transaction logic, that most obviously provide a model-theoretic semantics for state-transition systems, such as the situation calculus and MetateM, employ frame axioms in the operational semantics. We believe that the ability to update states destructively, without the use of frame axioms, is another important feature of LPS, which is essential for most practical applications. However, the importance of avoiding the use of frame axioms does not seem to be generally appreciated, making it difficult in many cases to compare LPS with other systems. It seems to us that this lack of appreciation is due to a confusion between the representational and computational aspects of the problem.

The *representation aspect of the frame problem* is how to represent formally the property that if a fluent holds in a state, then it continues to hold in future states unless and until it is terminated by an event. Arguably, this aspect of the frame problem has

been solved adequately by the use of frame axioms formulated in various non-monotonic logics, including logic programming.

However, it is easy to see that the *computational aspect of the frame problem* of efficiently determining whether a fluent actually holds in a given state cannot be solved in the general case by reasoning with frame axioms. It simply is not computationally feasible either to reason forwards with frame axioms, duplicating facts that hold from one state to the next, or to reason backwards, to determine whether a fact holds in a state by determining whether it held in previous states. The solution of the computational problem requires the use of destructive updates.

LPS/KELPS reconciles the use of destructive updates in the operational semantics with the model-theoretic semantics by making the frame axiom(s) emergent properties, which hold without the need to reason with them operationally.

10 Future Work

The operational semantics sketched in this paper is very abstract, and capable of many refinements and optimizations. We have implemented a prototype of LPS that includes some of these improvements, making it closer to a conventional programming language. For example, the implementation uses a Prolog-like depth-first search to choose a singleton set of disjuncts D . Some obvious additional improvements include the use of a constraint solver for handling temporal constraints and the use of a UML-like graphical external syntax.

LPS has its origins in AI knowledge representation and reasoning languages, but has developed into a framework that overlaps with many other areas of computing, including, most notably, database systems, as well as coordination languages that use a shared global state for parallel, concurrent and distributed computing. It is a major challenge to investigate the extent to which LPS might contribute to these areas.

Acknowledgements. We are grateful Harold Boley for helpful comments on an earlier draft of this paper and to Imperial College for EPSRC Pathways to Impact funding, which is supporting the implementation of LPS.

References

1. Bonner, Kifer, M.: Transaction logic programming. In: Warren, D.S. (ed.) Proc. of the 10th International Conf. on Logic Programming, pp. 257–279 (1993)
2. Carriero, N., Gelernter, D.: Linda in Context. Communications of the ACM 32(4) (1989)
3. van Emden, M., Kowalski, R.: The Semantics of Predicate Logic as a Programming Language. JACM 23(4), 733–742 (1976)
4. Fisher, M.: A Survey of Concurrent METATEM - The Language and its Applications. In: Gabbay, D.M., Ohlbach, H.J. (eds.) ICTL 1994. LNCS, vol. 827, pp. 480–505. Springer, Heidelberg (1994)
5. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (1982)

6. Fung, T.H., Kowalski, R.: The IFF Proof Procedure for Abductive Logic Programming. *J. of Logic Programming* (1997)
7. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Programming* 8, 231–274 (1987)
8. Harel, D.: Statecharts in the Making: A Personal Account. In: *Proc. 3rd ACM SIGPLAN History of Programming Languages Conference, HOPL III* (2007)
9. Hausmann, S., Scherr, M., Bry, F.: *Complex Actions for Event Processing*, Research Report, Institute for Informatics, University of Munich (2012)
10. Hayes-Roth, B.: A Blackboard Architecture for Control. *Artificial Intelligence* 26(3), 251–321 (1985)
11. Kakas, T., Kowalski, R., Toni, F.: The Role of Logic Programming in Abduction. In: *Handbook of Logic in Artificial Intelligence and Programming*, vol. 5, pp. 235–324. Oxford University Press (1998)
12. Knuth, D.E.: *The Art of Computer Programming*, vol. 1: Fundamental Algorithms. Addison-Wesley (1973)
13. Kowalski, R.: *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge University Press (2011)
14. Kowalski, R., Sadri, F.: From Logic Programming Towards Multi-agent Systems. *Annals of Mathematics and Artificial Intelligence* 25, 391–419 (1999)
15. Kowalski, R., Sadri, F.: Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents. In: Polleres, A., Swift, T. (eds.) RR 2009. LNCS, vol. 5837, pp. 1–23. Springer, Heidelberg (2009)
16. Kowalski, R., Sadri, F.: An Agent Language with Destructive Assignment and Model-Theoretic Semantics. In: Dix, J., Leite, J., Governatori, G., Jamroga, W. (eds.) CLIMA XI. LNCS, vol. 6245, pp. 200–218. Springer, Heidelberg (2010)
17. Kowalski, R., Sadri, F.: Abductive Logic Programming Agents with Destructive Databases. *Annals of Mathematics and Artificial Intelligence* 62(1), 129–158 (2011)
18. Kowalski, R., Sadri, F.: *Programming With Logic Without logic Programming*. Department of Computing, Imperial College London Report (2012)
19. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. *New Generation Computing* 4(1), 67–95 (1986)
20. McCarthy, J., Hayes, P.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: *Machine Intelligence*, vol. 4, pp. 463–502 (1969)
21. Przymusiński, T.: On the Declarative Semantics of Stratified Deductive Databases and Logic Programs. In: Minker, J. (ed.) *Foundations of Deductive Databases and Logic Programming*, pp. 193–216. Morgan Kaufmann (1987)
22. Reisig, W.: The Expressive Power of Abstract-State Machines. *Computing and Informatics* 22, 209–219 (2003)

Ontology-Based Query Answering with Existential Rules

Marie-Laure Mugnier

University of Montpellier
LIRMM / INRIA
France

Abstract. It is widely acknowledged that modern information systems require an ontological layer on top of data, associated with advanced reasoning mechanisms able to exploit the semantics encoded in ontologies. We focus here on ontology-based data access (OBDA), a new paradigm that seeks to take ontological knowledge into account when querying data. This paradigm is currently the subject of intense research in the database, knowledge representation and reasoning, and Semantic Web communities. Indeed, it is expected to have a major impact in many application domains, however some foundational issues need first to be addressed. In this context, we consider an emerging logical framework based on existential rules, also known as Datalog $_{\exists}$. This framework can also be defined in graph terms. Compared to the lightweight description logics currently developed for OBDA, it is more powerful and flexible; an important feature is that predicate arity is not restricted, which allows for a natural coupling with database schemas and facilitates the integration of additional information, such as contextual knowledge. On the other hand, the existential rule framework extends the deductive database language Datalog by enabling to infer the existence of entities that do not necessarily occur in the database (hence the name existential rules), a feature that has been recognized as crucial in the context of incomplete information. In this talk, we will provide an introduction to this framework in the context of OBDA, then present the main decidability and complexity results as well as algorithmic techniques, and discuss some challenging research issues.

A Model Driven Reverse Engineering Framework for Extracting Business Rules Out of a Java Application

Valerio Cosentino^{1,2}, Jordi Cabot¹,
Patrick Albert², Philippe Bauquel³, and Jacques Perronnet³

¹ AtlanMod, INRIA & EMN, Nantes, France

² IBM CAS France

³ IBM, France

{albertpa,bauquel.p,jacques_perronnet,valerio.cosentino}@fr.ibm.com,
jordi.cabot@mines-nantes.fr

Abstract. In order to react to the ever-changing market, every organization needs to periodically reevaluate and evolve its company policies. These policies must be enforced by its Information System (IS) by means of a set of business rules that drive the system behavior and data. Clearly, policies and rules must be aligned at all times but unfortunately this is a challenging task. In most ISs implementation of business rules is scattered among the code so appropriate techniques must be provided for the discovery and evolution of evolving business rules.

In this paper we describe a model driven reverse engineering framework aiming at extracting business rules out of Java source code. The use of modeling techniques facilitate the representation of the rules at a higher-abstraction level which enables stakeholders to understand and manipulate them.

1 Introduction

Today market needs oblige organizations to change periodically their policies expressed as a set of business rules. A business rule represents a relevant action or procedure aiming at defining or constraining some precise aspect of a business. Business rules are a key component of the Information System (IS) of the company. Unfortunately, they are not usually implemented as a single and easily identifiable component in the IS but are generally scattered in many parts of the IS source code. This makes it very difficult to quickly and safely evolve the organizational policies.

To tackle this issue we propose a new Business Rule Extraction (BREX) framework. BREX [1] is the process of extracting business rules out of an IS, isolating the code segments which are directly related to business processes. BREX includes three major activities: Variable Classification, Business Rule Identification (mainly based on Program Slicing [2] techniques) and Business Rule Representation.

Variable Classification is used to reduce the number of variables to analyse. It aims at finding variables that represent domain/business concepts and hint at business rules. The set of domain concepts represent the sphere of non-technical knowledge embedded in the application.

Business Rule Identification aims at identifying business rules by slicing the source code [3] to focus on the chunks of code that are relevant to the domain variables, identified in the previous step. A set of chunks related to the same variable conforms a business rule.

Business Rule Representation consists of presenting the extracted business rules through artifacts (graphs, text, ...) amenable to human comprehension.

A further activity in BREX is the traceability of business rules from the source code. This task is not always present in BREX frameworks but we believe is a key component to explain and justify the origin of the extracted business rules.

In this sense, this paper describes a model-driven framework for extracting business rules out of a Java application. We show that Model Driven Engineering (MDE) applied to reverse engineering/BREX approaches offers some important benefits with respect to previous works. MDE allows working on an abstract homogeneous representation of the system that avoids technological problems and provides a non-intrusive solution, since the extraction process is performed by working with the model of the system and not the system itself. Moreover, when representing the system as a model we can benefit from the plethora of MDE tools available to manipulate the (model of the) system.

MDE allows us to build a framework composed by independent and modular steps, since each of them is related to the others by its input and output models. In this way, the user can stop the BREX process at any moment, selecting the level of output that better fits his needs.

The framework is fully automatic but allows user intervention at the end of each sub-step. This allows users to complement the automatic process in order to refine and improve the results of our extraction heuristics, e.g. users could provide information about the company "coding style" to facilitate the identification of rules.

This paper is structured as follows: Section 2 presents a running example; Section 3 introduces the overall approach; Section 4 illustrates Traceability in the framework, which describes how the entities composing the artifacts in the framework are related to the source code; Section 5 analyses the result of this framework; Section 6 discusses the related work and Section 7 closes the paper with conclusion and future work.

2 Running Example

In order to illustrate our framework, we will use as running example a Java application that belongs to the simulation software category and that contains several business rules.

The application simulates the behavior of animals and humans in a meadow, where each actor, animal or human, can act and move according to its nature. Two different functionalities are implemented in this application: one represents the business logic and describes how predator-prey interactions affect population sizes. The second one is used to store statistical information about the actors participating in the simulation.

A schema of the application classes and their relationships is shown in Fig.1. *GUI* class shows the graphical interface of the application; *Simulator* simulates the predator-prey game and it stores information for statistical analysis; *SimulatorView*, *AnimatedView* and *FieldView* represent the graphical views of the game. *Counter* provides a counter for each participant in the simulation; *Grass* models the grass on the field; *Field* is a rectangular grid of field positions. Each position is modelled as a *Location*. *Actor* is an interface containing methods to modify the actor's location and to perform the actor's daily behavior. *Human* and *Animal* implement *Actor*. *Human* provides the common features to all humans (get/set location). *Animal* stores the actual *age*, the *location* in the field and the *food level*. It contains also a boolean variable for determining if the animal is *alive*, the maximum and the breeding age, the breeding likelihood and the maximum number of births which an animal can have.

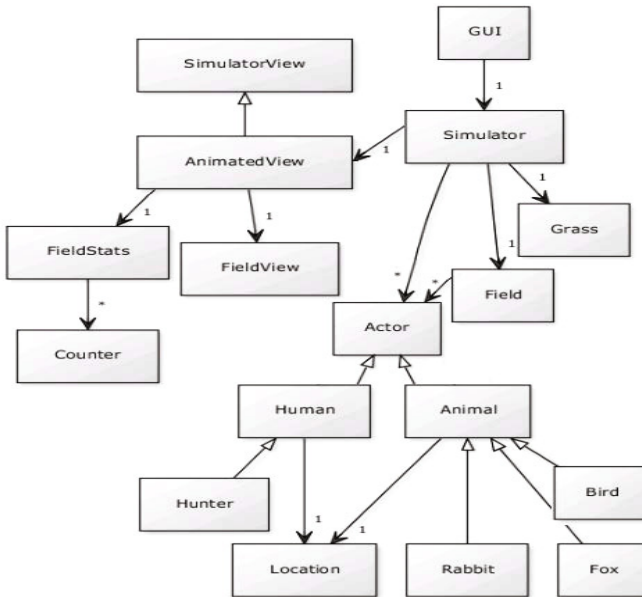


Fig. 1. Class dependencies

2.1 Rules Modeling the Application

A manual inspection of the source code of these classes reveals the existence of several business rules.

Rules modelling hunter behaviours are:

- Hunters never die
- Hunters hunt animals

Rules modelling bird and rabbit behaviors are:

- Rabbits/Birds can die by being eaten by foxes, hunted by hunters, because of starvation, old age or overcrowding
- Rabbits/Birds can breed when they reach their breeding age
- Rabbits/Birds eat grass

Rules modelling the fox behaviors are:

- Foxes can die by being eaten by hunters, because of starvation, old age or overcrowding
- Foxes can breed when they reach their breeding age
- Foxes eat rabbits or birds

Fig. 1 shows also the inheritance rules, but to detect them it's not necessary to perform an analysis like the one presented in this paper. The application is composed by 2 packages and 16 classes. The presentation and the domain layers are clearly separated.

3 Framework Description

As written in Section 1, a BREX is typically composed of three operations: Variable Classification, Business Rule Identification and Business Rule Representation. A new operation, Model Discovery, is added to the framework in order to move the global BREX process from a grammarware technological space to the modelware one. Fig. 2 depicts these four phases together with the input/output artifacts of each phase.

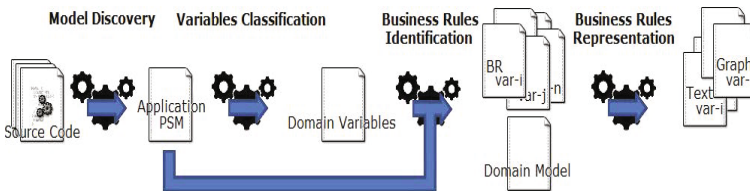


Fig. 2. Overall approach

Model Discovery takes as input the source code of a Java application and generates a Java model that has a one-to-one correspondence with the code (i.e. there is no information loss; all classes, methods, behavior,... of the Java code is represented as part of the model). We will refer to this Java model as Platform Specific Model (PSM) in the remainder of the paper since model discoveries are available for several languages and could be reused in other BREX processes.

Variables Classification identifies the domain variables together with their containing classes. The input of this operation is the PSM and the output is a model containing all domain's classes and their inner variables.

Business Rule Identification provides the means to identify the business rules related to a domain variable. This operation takes as input the PSM and a variable i contained in the Domain Variable Model. It returns two models: a model containing the internal representation of the business rules belonging to i and a global domain model with the set of classes, method signatures and class attributes relevant for the union of domain variables models.

Business Rule Representation provides artifacts for representing business rules. This operation takes as input the Business Rule Model for the variable i and returns human-understandable artifacts that ease the comprehension of the business rules for i .

The model discovery phase is implemented with MoDisco [4]. MoDisco is a tool offering a set of model-based components to facilitate the creation of reverse engineering solutions. MoDisco includes already a Java metamodel and a full Java discovery that instantiates this Java metamodel with based on the source code of a set of Java files.

The other three phases, which are the ones strictly corresponding to a BREX process, are described in detail in the next subsections. They have all been implemented by means of a chain of model-to-model transformations that manipulate the input and output models as described in the text. All transformations have been implemented using Atlas Transformation Language (ATL) [5], which is a model transformation language specified as both a metamodel and a textual concrete syntax.

In the field of MDE, ATL provides developers with a means to specify the way to produce a number of target models from a set of source models by writing rules that define how to create target models from source model elements.

3.1 Variable Classification

Variable Classification is used to reduce the number of variables to analyse by filtering out those variables which are not representing (or relevant for) domain information. This phase takes as input the PSM and returns a model with the Java classes and variables modeling business concepts. These variables are used as starting point to identify business rules.

To identify the relevant variables we have developed a set of heuristics based on a sample of Java programs. For instance, for the running example, the heuristics help to distinguish between classes belonging to the business layer and classes

belonging to the presentation layer based on the package and import directives in the class definition.

All classes in the presentation layer are collected and used as starting point to find classes handling domain concepts. Since several functionalities can be implemented in an application, the domain classes are organized in groups. The classes composing a group contain one or more type dependencies of other classes in the same group. Groups having the same classes are merged together.

The computation of calculating a group starts by creating the set of classes using graphical imports (*GUI*, *Simulator* and *AnimatedView*, *FieldView*; while *SimulatorView* is not considered because it is an interface). From each of these classes three lists are generated: a list (output) containing the classes already analysed, a list (temp) containing the classes which have a type dependency to the current analysed class and a static list (forbidden) of classes that can not be part of the group. The computation ends when the temp list is empty.

The variables in the classes of the output list are classified in three categories: single-access, multi-access and potentials.

- *Single-access variables* are all the class attributes that occur at most once on the left side of an assignment. In this group we can find final and static variables and variables that are initialized in the constructor.
- *Multi-access variables* are all the class attributes occurring more than once on the left side of an assignment.
- *Potential variables* are all the variables that are declared in methods and occur on the left side of an assignment.

Single-access variables point at business rules modelling the initializations of an application; whereas *Multi-access* and *Potential* variables point at business rules modelling more complex behaviors.

The metamodel in Fig 3 is used to store the variables information. The metamodel is composed by a root entity *Model* containing zero or more groups. Each *Group* stores a set of *classes* related to it. A *Class* is a subtype of *Element*.

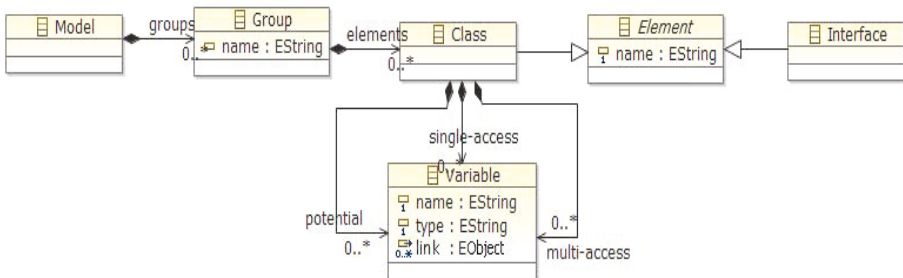


Fig. 3. Variable Classification metamodel

Class is described by three lists of *Variables*: single-access, multi-access and potentials. *Variable* is described by three properties: *name*, storing the name of the variable; *type*, storing the type name of the variable and *link*. The latter is used to store a reference to the entity in the PSM that corresponds to the variable declaration statement in the code.

3.2 Business Rule Identification

Business Rule Identification, described in Fig. 4, is composed of several sub-steps: Domain Model Extraction, Slicing Operation and Business Rule Model Extraction. It takes as input the PSM model and the Domain Variables Model and generates two models (Domain Model and PSM enriched with slicing annotations (PSMA)). The first one stores a map between the domain concepts expressed as class names, method signatures and class attributes pointed by the domain variables and a customizable verbalization of these elements (to improve the quality of the natural language explanation of the rules); whereas the second one contains all the business rules related to a domain variable *i* selected by the user.

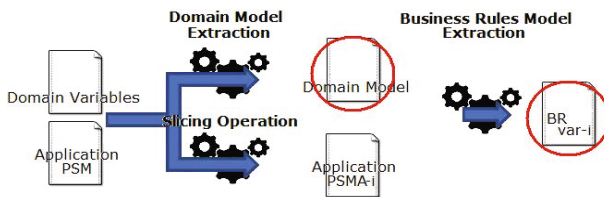


Fig. 4. Business Rule Identification process

Domain Model Extraction. This operation allows extracting method signatures and class attributes from the classes containing the domain variables identified in the variables classification step, providing a default vocabulary for these entities to be reused in the description of the business rules. The default verbalization consists in simply splitting the names of classes, variables and methods according to the common way to define them in Java (for example, for static and final method or variable names: ABC_DEF ->ABC DEF; in the other cases: abcDef ->abc Def). Nevertheless, the user can tune the process and define its own rule verbalization (or directly change the verbalization of some methods).

The input of this operation is the PSM model and the Domain Variables Model. The output of this step is a model conforming to the Business Object Model/Vocabulary Model (BOM/VOC) metamodel of IBM WebSphere ILOG JRules BRMS¹.

¹ <http://www-01.ibm.com/software/integration/business-rule-management/jrules-family/>

In Fig. 5, a part of the BOM/VOC model concerning the class *Grass* is shown. All the method signatures and class attributes belonging to *Grass* are stored in a model conforming to the BOM metamodel. This model is used to generate an instance of the VOC metamodel containing a default verbalization for all the BOM elements. The name of the class is translated as a *concept*, while variables and method signatures are translated as *phrase* in which the word *this* is used to refer to *concept*.

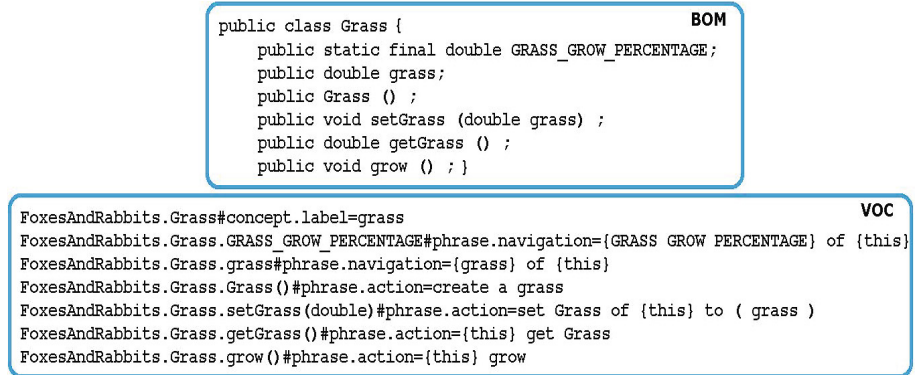


Fig. 5. Example of the BOM/VOC model for the class *Grass*

Slicing Operation. The slicing operation is a variation of block slicing [6]. The inputs of this step are the PSM and a variable i contained in the Domain Variables Model; whereas the output is the PSM enriched with annotations (PSMA) on all the statements, variable declarations and methods relevant for i . Each annotation for any of those elements concerns the granularity index, the name of the slicing variable, the unique rule number and the type of relation with the slicing variable i .

The granularity index is the position of a method (containing one of the elements relevant to i) inside the ordered set of methods we cross in a program from the main entry execution point to the statement that actually modifies the value of the variable i . This ordered set of methods is defined as *granularity set*.

A relevant statement can be annotated as *rule* or *related*. All the statements that allow passing from a method in the *granularity set* to another one in the same set are annotated as *rule*. A statement is marked as *related* if it contains a *rule* statement or contains a variable declaration used inside a *related* or *rule* statement.

Two types of relations are defined for variable declarations. A variable declaration is marked as *sliced-variable* if it is the selected slicing variable i . A variable declaration is marked as *related-variable* if it is used inside a *related* or *rule* statement.

Relevant methods can be annotated as *related* if they contain at least one *related* or *rule* statement or as *reachable* if one of its invocations occurs in a *related* statement or in another *reachable* method.

All this information is then used to extract the business rule. The result of the annotation can be visualized by the user if desired. The previously mentioned MoDisco Eclipse plug-in can take the annotated model and transform it back into a Java application where all annotations will appear as comments.

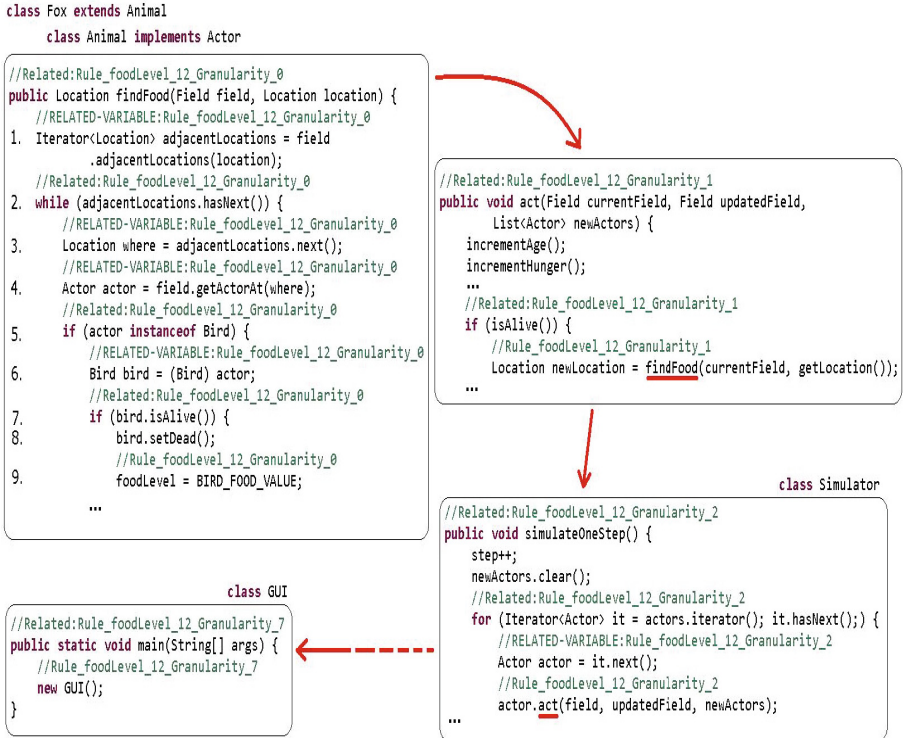


Fig. 6. Example of a slicing operation on foodLevel variable

In Fig. 6 an example of slicing is presented. Line 9 contains the *rule* statement of the slicing variable *foodLevel* for granularity zero. The if condition at line 7 is annotated as *related* since it contains the *rule* statement. The statement at line 6 is annotated as *related* because the statement at line 7 cannot exist without it. The statements at line 5 and 4 follow the same logic. The variable declaration statement at line 3 is annotated as *related*, since the defined variable is used as argument at line 4. The while statement at line 2 is marked as *related*, because it contains statements that are related to the slicing variable. The statement at line 1 is annotated as *related*, because it is used in the condition of the next

while statement. The method is marked as *related*, since it contains *related* and *rule* statements.

The method *findfood* is invoked from the body of the method *act*, which contains elements related to the slicing variable with granularity 1. The two methods are in the same class *Fox*.

The method *act* is invoked in *simulateOneStep* of the class *Simulator*. The body of this method contains elements with a granularity value of 2.

Analysing where the *related* methods are invoked, it is possible to go back until the method that starts the application.

Business Rule Model Extraction. The goal of this step is to extract from the PSMA only those entities that are annotated and domain-related to the variable *i*.

As seen in Fig. 6, the slicing operation allows tracking all the methods and statements for a specific domain variable. Since a part of those methods are outside the domain layer, we use the information collected during the Domain Model Extraction step to identify and remove them.

The input of this transformation is the PSMA and the Domain Model; the output model contains all the business rules for the variable *i*. Each business rule contains statements, methods and variable declarations annotated during the slicing operation that have references to the domain concepts stored in the Domain Model.

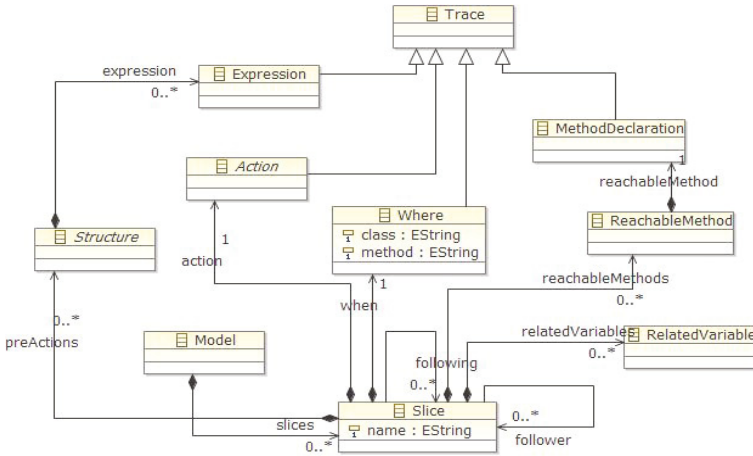


Fig. 7. Business Rule metamodel

The output model conforms to the metamodel shown in Fig. 7. The *Where* entity stores the class and method names from which the *Slice* has been extracted. An *Action* entity represents a *rule* statement that can be a method invocation, an assignment, an object creation statement or a variable declaration. The pre-Actions list contains *Structures* related to the *Action*. A *Structure* can be a loop

statement, a variable declaration or an if statement. Each *Structure* can store zero or more *Expressions*. *ReachableMethod* and *RelatedVariable* entities contain the methods and the variables that are invoked in *Structures* and *Action*. *Expression*, *Action*, *Where* and *MethodDeclaration* are *Trace* entities used to store links pointing to the PSMA elements from which they have been generated.

Slice entities are related each other by following and follower list, that allow creating a graph of slices. For each slice s , the first list contains slices which store *rule* statements having the same id number of s and the immediately superior granularity index. The second list contains slices storing *rule* statements having the same id number of s and the immediately lower granularity index.

3.3 Business Rule Representation

Business Rule Representation, shown in Fig. 8 provides human-understandable artifacts describing the extracted business rules for the slicing variable i .

This process takes as input the Business Rule Model for the variable i and optionally the Domain Model. The latter is used if the user wants a verbalization not completely based on the source code. It generates textual and graph artifacts for easing the analysis of the extracted business rules.

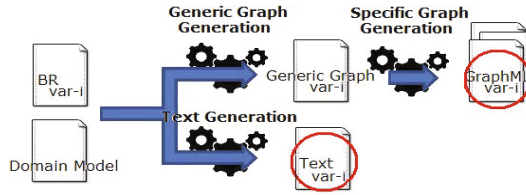


Fig. 8. Business Rule Representation process

Text Generation. The text generation takes as input the Business Rule Model and the Domain Model if selected. It generates a textual output where the sentences contain the verbalization of the entities stored in the Business Rule Model (Fig. 10).

Generic Graph Generation. Since several types of graph exist and since each of them can be used to emphasize some topology features; the framework allows transforming the Business Rule Model into a generic graph model, that collects edges, nodes, their labels and dependencies.

This step takes as input the Business Rule Model and the Domain Model and generates a model conforming to the Portolan metamodel [7], that allows bridging the gap between data of a given domain and its graph visualization.

Specific Graph Generation. Thanks to Portolan [8], we can delegate the selection of a particular type of graph to a dedicated step.

This step takes as input the Portolan model and produces as output a specific graph model, which currently conforms to a metamodel representing a GraphML graph².

4 Traceability Support in the Framework

Traceability in BREX can be defined as the ability to tie the source code elements to those composing the extracted business rule [9]. Our approach offers full traceability support between all the steps.

Our traceability implementation benefits from the key importance of the traceability concept in MDE where generation of traces is already part of the features offered by several model manipulation tools (e.g. in transformations [10], to relate the target elements with the source elements that originated them; see [11] for a survey of traceability approaches in MDE).

Given that our framework is MDE-based, we can implement BREX Traceability through MDE Traceability using non-intrusive methods. This is an important difference with respect to other methods that must use more intrusive actions (e.g. modifying the compiler to instrument the code to generate traces) to collect the needed information.

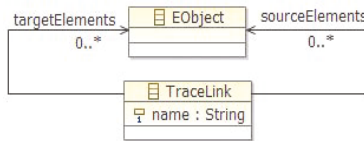


Fig. 9. Traceability Metamodel

Traceability information is stored in a traceability model conforming to the trivial metamodel of Fig. 9). *Traceability* entity stores the sets of linked source and target elements (generic *EObjects*) for all the rules executed in the ATL transformations implementing the different steps of our method. Therefore, each transformation rule creates not only the elements of the target model but also links each target element with the source element that matched the rule and triggered its execution.

5 Analysis of the Result

To validate our method we analyzed that the business rules returned at the end of the BREX process for the running example coincide with the ones that we

² <http://graphml.graphdrawing.org/>

discovered after a manual inspection. For the running example, we were able to generate both graphical and textual representations of all the identified rules, facilitating this way the comprehension of the application.

As an example of the results obtained on the application described in Section 2, we show some of the extracted business rules.

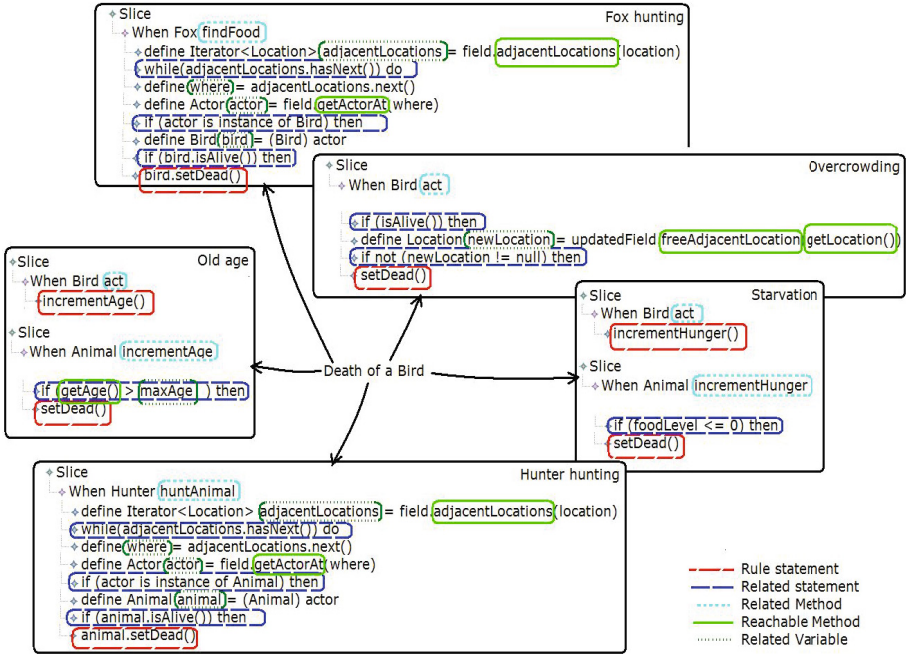


Fig. 10. Causes of death of a bird

Fig 10 presents a textual-based explanation of all possible causes of death for birds. Each box is automatically generated and summarizes a different business rule controlling the birds’ death: a bird can die hunted by a fox or by a hunter, because of starvation, old age and overcrowding.

Currently we are testing our framework on a bigger case study provided by IBM, but due to lack of space we can report only a part of the new result.

Thanks to our approach we have been able to discover uncovered rules that the users were not aware of.

The IBM case study has allowed us to analyse the efficiency of our framework. The most time-consuming step is the Slicing Operation (Section 3.2), since it is based on recursive heuristics, which identify the relevant input elements for a given variable and write annotations on them.

In order to optimize this step, we are implementing a pruning component that will allow reducing the input size of the slicing operation for any given variable.

We have remarked that the expressiveness of the inferred rules decreases as long as the complexity of the application domain increases. In the example described in this paper, the default verbalization allows going up towards a language that is not programming-related. Unfortunately, this does not happen for the complex case study, where the default verbalization adds more complexity to the rule expressiveness.

6 Related Work

BREX has been extensively studied in the literature but we believe our approach provides some additional benefits with respect to previous work.

First of all, the output of the framework is flexible. Thanks to the modularity provided by the use of MDE techniques, we can separate the internal representation of the rules from their external visualization. This separation makes it possible to create different verbalizations for the same business rule. In previous work like [1], [12], [13] and [14] the verbalization step and a separation between the internal and the external representations are not provided.

Traceability is also missing in most of the approaches [12], [13], [1], [15]. [16] includes partial traceability support implemented by means of adding start line number, end line number and annotations to the business process that facilitate identifying the parts of the code relevant to the process. Instead, thanks to the explicit relationships between the business rule model and the Java model, we can navigate from one to the other and retrieve the exact code excerpt relevant to the rule.

Regarding approaches specific for Java, [17] proposes an intrusive approach based on the byte-code instrumentation. Our approach is non-intrusive, since we work on an abstraction of the system.

In all of those papers the Granularity of the extracted business rules is not treated or mentioned.

7 Conclusion and Future Work

This paper describes a MDE framework for extracting BRs out of a Java applications. The BRs extracted out of the source code are stored in a model-based internal representation that can be externalized in several ways to fulfill the needs of different users (business analysts, developers, ...). Moreover, our integrated traceability mechanism allows to link back the rules to the corresponding part of the source code that justifies their extraction.

The four steps composing the framework have been explained at high description level, since we have preferred to discuss their heuristics, their input and output instead of entering in details for each of them.

The example used along this article has been selected in order to develop a framework that could be used for understanding a generic application.

We are now applying our framework on a real use case provided by IBM and composed by more than 5000 Java classes. This will help us to develop additional

heuristics for the framework and test its scalability. Moreover, we plan to extend the framework to other technologies beyond Java. In particular, we will focus our attention to the identification and consolidation of business rules enforced as part of the presentation and persistence (e.g. as part of checking conditions in triggers) layers. Finally, we would like to integrate machine-learning capabilities so that the framework becomes able to learn both about the coding and implementation style used by the company (so that the heuristics can be refined based on the corrections provided by the users in previous projects) and about the domain itself (i.e. the business rules extracted for the domain can be used as auxiliary information when extracting business rules of another software for the same domain).

References

1. Sneed, H.M., Erdős, K.: Extracting business rules from source code. In: WPC, pp. 240–247 (1996)
2. Weiser, M.: Program slicing. *IEEE Trans. Software Eng.* 10(4), 352–357 (1984)
3. Tip, F.: A survey of program slicing techniques. *Journal of Progr. Lang.* 3(3), 121–189 (1995)
4. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: Modisco: a generic and extensible framework for model driven reverse engineering. In: ASE, pp. 173–174 (2010)
5. Link: Atlas transformation language, <http://www.eclipse.org/at1>
6. Korel, B., Yalamanchili, S.: Forward computation of dynamic program slices. In: ISSTA, pp. 66–79 (1994)
7. Mahe, V., Martinez Perez, S., Doux, G., Brunelière, H., Cabot, J.: Portolan: a model-driven cartography framework. Technical Report RR-7542 (2011)
8. Link: Portolan, <http://code.google.com/a/eclipseorg/p/portolan/>
9. Baxter, I., Hendryx, S.: A standards-based approach to extracting business rules, <http://www.semdesigns.com/Company/Publications/ExtractingBusinessRules.pdf>
10. Jouault, F.: Loosely coupled traceability for atl. In: ECMDA, pp. 29–37 (2005)
11. Galvão, I., Goknil, A.: Survey of traceability approaches in model-driven engineering. In: EDOC, pp. 313–326 (2007)
12. Huang, H., Tsai, W.T., Bhattacharya, S., Chen, X.P., Wang, Y., Sun, J.: Business rule extraction from legacy code. In: COMPSAC, pp. 162–167 (1996)
13. Putrycz, E., Kark, A.W.: Recovering Business Rules from Legacy Source Code for System Modernization. In: Paschke, A., Biletskiy, Y. (eds.) RuleML 2007. LNCS, vol. 4824, pp. 107–118. Springer, Heidelberg (2007)
14. Barbier, F., Deltombe, G., Parisy, O., Youbi, K.: Model driven reverse engineering: Increasing legacy technology independence. In: IWRE (2011)
15. Fu, G., Shao, J., Embury, S.M., Gray, W.A., Liu, X.: A framework for business rule presentation. In: DEXA, pp. 922–926 (2001)
16. Zou, Y., Lau, T., Kontogiannis, K., Tong, T., McKegney, R.: Model-driven business process recovery. In: WCRE, pp. 224–233 (2004)
17. Felix Lösch, J.L., Schmidberger, R.: Instrumentation of java program code for control flow analysis (2004)

Business Process Data Compliance

Mustafa Hashmi^{1,2}, Guido Governatori^{1,2}, and Moe Thandar Wynn^{1,2}

¹ NICTA, Queensland Research Laboratory, St. Lucia Australia
{mustafa.hashmi, guido.governatori}@nicta.com.au

² Queensland University of Technology (QUT) Brisbane, Australia
m.wynn@qut.edu.au

Abstract. Most approaches to business process compliance are restricted to the analysis of the structure of processes. It has been argued that full regulatory compliance requires information on not only the structure of processes but also on what the tasks in a process do. To this end Governatori and Sadiq [2007] proposed to extend business processes with semantic annotations. We propose a methodology to automatically extract one kind of such annotations; in particular the annotations related to the data schema and templates linked to the various tasks in a business process.

Keywords: Business process, business process compliance, database schema, compliance by design.

1 Introduction

Recently much interest has been seen in the business process management community on business process compliance due to the introduction of new regulatory laws such as Sarbanes-Oxley, BASEL II, and HIPPA to name a few. These laws impose severe penalties on violations. Hence enterprises are heavily investing to comply with internal or external policies thus the compliance software and services industry is booming rapidly. A recent survey by Deloitte Australia [1] reveals that, in Australia alone, estimated spending on compliance related activities in the public sector is around 4% of total IT spending (approaching the annual cost of AUD\$1-billion), and compliance costs are expected to rise in coming years. No matter what they have to do, enterprises are obliged to streamline their daily business operations to the regulatory laws for transparency and better operations.

Enterprises develop process models to document and automate their operational activities. These process models provide an enterprise with a high-level view on how to achieve their business objectives and implement regulatory policies governing these processes. Hence process models can be used to verify the effectiveness of regulatory laws and/or policy controls. Furthermore, these models can also provide a view on the flow of data and relationships among the activities in the process, thus making a process model a natural venue to implement compliance related controls. Essentially compliance is a relationship between two distinct spaces with different objectives: *process modeling specifications space* and *business rules specifications space*. The business modeling specification space is procedural in nature, detailing how a business activity

should take place. In contrast, business rules specifications are descriptive, dictating what need to be done to remain compliant [2].

Achieving balance between these two different worlds is not straightforward as a number of efforts have been reported in business process management literature [3–5]. However, predominantly much of these efforts have been limited to the development of descriptive approaches for BPM to achieve flexibility in business process execution [6] or restricted their attention to the analysis of the structure of business processes only [7, 8]. As compliance requirements come from different sources, it has been argued that to achieve full compliance it is inevitable to have complete information not only on the structure of processes, but also on what the tasks in a process do. To this end [2] proposed to enrich business processes with semantic annotations. Enhancing processes with these annotations allows process designers to implement, and see the control objectives within the process modeling space.

The idea to semantically annotate business processes is based on the notion of control tags. Control tags provide better understanding of the interaction between business process modeling specifications and business rule specifications. From a business process model perspective, there are four types of control tags: *control-flow*, *data*, *resources*, and *time* control tags [2]. These control tags consist of the state and operations of propositions about the conditions that are to be checked on a task; and are typed linked. In addition to that, control tags are not based on specific ontology, and may have associated constraints or policies. We build our work on the idea of these control tags with primary focus on the data control tags which identify the data retention and lineage requirements. For compliance checking purpose, the data control tags can be designed through parsing of Formal Contract Language¹(FCL) expressions, representing business rules. However, the problem is that how do we get the data for the data control tags; and where the data will come from. In addition to that, another question is how we can enforce the data constraints when annotating a process model with data tags. In this paper we are interested with the first two questions only: how to get data for data control tags; and from where. To address this problem, we proposed a query-based methodology to extract the data for control tags to annotate process models.

Business rules can be used for a variety of purposes. One particular application of business rules is to capture constraints on the data used in/by an application. Business rules provide a declarative approach to model such constraints and typically, they do not force specific technology and implementations.

A business process is a self-contained description of the activities to be done to achieve particular business objectives, the order in which the activities have to be done, the data the process operates on, and the resources required by the process. In this paper we restrict our focus on the data aspect. Figure 1 shows the links between the data and a task of a process model. Typically, there are three possible ways to provide data to a task in a process: (i) *the tasks receives the data from a previous task*, (ii) *the task is given data from a user*, and (iii) *the task reads data from a database linked to the process*. We can reverse the direction for the data produced by a task. In general for the data interactions we assume that: (a) *users interact with a process using forms and get reports back following some specific templates*, and (b) *data passes from one task to*

¹ FCL: A formalism to express normative specifications.

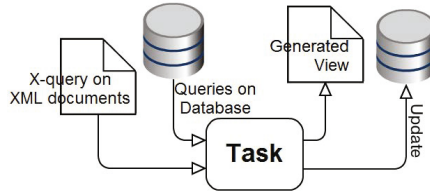


Fig. 1. Links between Data and Task in Process Model

the other using messages (according to specified templates or schemas). Thus data can be obtained by querying a database or parsing (XML) documents. The data produced, again is obtained by queries on databases (including the generation of views).

A typical scenario for the methodology we are going to discuss in the rest of paper is that where we have document-centric business processes, and where there is an organization that provides requirements for what data has to be in the documents, and how the documents are handled. The issue is to provide compliance certification for a process implementing the business rules specifications. A prototypical example is that of electronic lodgment of applications (for which we provide a simplified scenario based on a real life case, in Section 2).

Since we focus on compliance (i.e., design-time verification of the alignment of two sets of specification), it is not possible to have the actual data of instances of a process. Accordingly, the data control tags are not about the data of an instance case of a process, but on the schema of the databases and the parameters of document templates. Thus the research question of the paper is: *how to extract relevant information from the schema of the databases linked to a process.*

The organization of the paper as follow: in Section 2, we introduce a motivating scenario to set the stage to present our methodology. A short discussion on business process compliance follows in Section 3. The basics of formal contract language (FCL) will be presented in Section 4 after a short discussion on modeling normative requirements. In Section 5, we show how FCL can be used to model the business rules of our motivating example. Section 6 will outline our proposed compliance by design schema extraction methodology, followed by a review of latest research in the problem domain in Section 7. Concluding remarks and an outlook on future work will be presented in Section 8.

2 Scenario: Lodgment Verification Process

To present our proposed methodology consider a hypothetically simple lodgment case verification process aiming to verify whether the lodgment case comply with all designated rules, and whether it is in an acceptable form for further processing (cf. Figure 2).

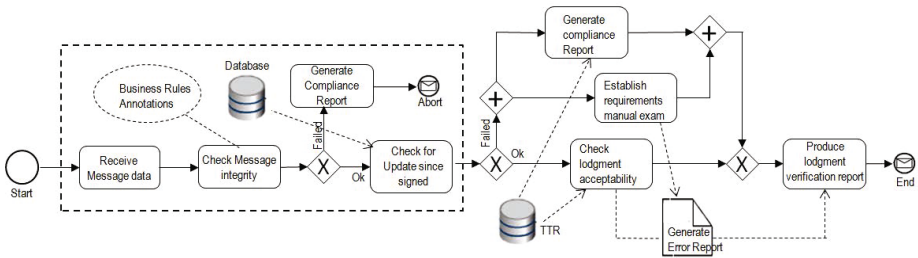


Fig. 2. Lodgment Case Process

The data verification process can generate the following response:

- an indication that the lodgment case meets all requirements for the lodgment, or
- a list of business rules with which that lodgment case does not comply, including registration requirements, and/or where required,
- a list of the manual examination processes that need to be performed on that lodgment case following the lodgment.

The process starts with a verification request message from the subscriber containing message data items included in the message header such as *electronic lodgment notice workspace ID*, *request message type*, *system request ID* (from where message originates), *operator ID*, *lodgment case ID* just to name a few. The application consists of four documents:

- Electronic Lodgment Case (*eLC*),
- eConsent Information Report (*eCIR*),
- eLodgment Information Report (*eLIR*), and
- eNotice of Sale Information Report (*eNoSIR*).

In addition to basic data in the verification request message data, additional information pertaining lodgment case must be present in these documents. The main objective of the verification process is to verify that these documents are present in an application, and that they contain the information required to verify the suitability of the lodgment case. Furthermore, the required data must be in the required format.

There are four documents associated with the request message data of the lodgment case process. The request message must contain data items from these documents. The data requirements are expressed by the following business rules.

Business Rules

- BR1 Each eConsent Information Report must specify exactly one ELN (Electronic Lodgment Notice) workspace ID.
- BR2 The ELN workspace ID specified in each eConsent Information Report must be the same as the ELN workspace ID specified in the eLodgment Information Report in the eLodgment Case that includes that eConsent Information Report.

- BR3 Each eConsent Information Report must specify exactly one ELN eLodgment Case ID.
- BR4 The ELN eLodgment Case ID specified in each eConsent Information Report must be the same as the ELN eLodgment Case ID specified in the eLodgment Information Report ID in the eLodgment Information Report, in the eLodgment Case that includes that eConsent Information Report.
- BR5 Each eNoS Information Report must specify exactly one ELN eLodgment Case ID.
- BR6 The ELN elodgment Case ID specified in each eNoS Information Report must be the same as the ELN eLodgment Case ID specified in the eLodgment Information Report included in the eLodgment Case that includes the eNoS Information Report.

In the next sections we introduce the methodology for business process compliance and the formalism we are going to use to formalize the above business rules.

3 Business Process Compliance

The main objective of business process compliance is to ensure that businesses perform their operations in accordance with regulatory laws and/or internal policies. Business rules on one hand, and business process modeling on the other, are two separate worlds with different objectives. The business rules specifications (a.k.a *normative specifications*) dictate what business has to do, in contrast, process modeling specifications describe how a business activity is performed. To properly verify that a business process is fully compliant with designated normative specifications, it is compulsory to provide a conceptually rich representation of both normative specifications and business process modeling specifications. This defines what obligations and permissions a business process is subject to. To capture the real intention of the business rules and for effective compliance checking, a formally rich representation of normative specifications is mandatory and challenging. We follow the methodology proposed by Governatori and Sadiq [9, 2, 10] for business process compliance. The key aspects of the methodology are: (1) *to enrich business process models with semantic annotations*, (2) *to extract control objectives from business rules*, and (3) *to formalize the control objectives in an appropriate logical formalism*.

Annotations can be at the level of a business process or at the level of tasks, where each task can have its own annotations. The annotations for a task, essentially, provide additional information about what the task does (effects to the task), the resources involved in a task, the data associated or produced by a task. The effects of the tasks are accumulated over the tasks in an execution trace of the process using an update semantics [11–13], and compliance is checked based on the algorithm proposed in [13, 14]. Figure 3 depicts the architecture of business process compliance. The proposed methodology goes well beyond simple structural compliance (i.e. checking the structure of a business process). The cost for this is to have semantic annotations and properly modeling normative requirements. Most of the semantic annotations must be given by domain experts. However, in the rest of the paper we are going to show how annotations for data constraints on document-centric business processes can be extracted automatically

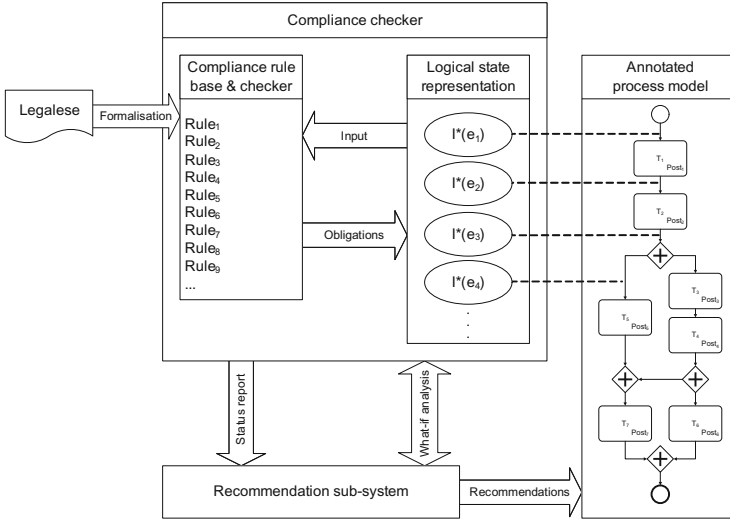


Fig. 3. Business Process Compliance Architecture: Adopted from [12]

from the schemas/templates associated to the process. In the next section we outline how the formalism (FCL) proposed in [9] can be used for properly modeling normative requirements.

4 Modeling Normative Requirements

Deontic logic studies formal properties of normative specifications in terms of the so called normative positions (i.e. obligations, permissions and prohibitions). Deontic logic provides machinery to investigate relationships among different normative positions. A detailed discussion on deontic logic and Standard Defeasible Logic (SDL) is beyond the scope of this paper, the reader is directed to [9] for further reading. The problem with deontic logic is that, it is not capable of dealing with violations and the obligations arising from violated obligations [15]: in some situations business rules may specify conditions about when other conditions in business rules books have been violated (i.e. some clauses of the rules have been violated). Deontic logic does not provide a faithful representation of such situations. Governatori [16] proposed Formal Contract Language (FCL), a formalism to analyze business contracts and address the deficiencies of deontic logic by providing rich representation of contract violations. We used FCL to provide formal representation of lodgment verification process. FCL is a rich combination of an efficient non-monotonic formalism (defeasible logic (cf. [17, 18]), and deontic logic of violations (Governatori and Rotolo [19]) which enables us to present exceptions as well as ability to capture violations. Moreover, FCL provides for a conceptually rich formalization of norms for compliance checking of a process where partial information and possibly conflicting provisions are present.

FCL consists of two sets of atomic symbols: a numerable set of propositional letters $a, b, c \dots$ that represent the state variables and the tasks of a process. Formulas of the

logic are built using the deontic operators O (for obligations), P (for permission), negation \neg and the non-structural connective \otimes (for the contrary to duty operator). An FCL formula is defined in a two-step process under the following formation rules:

- every propositional letter is a literal;
- the negation (\neg) of a literal is a literal;
- if X is deontic operator and l is a literal then Xl and $\neg Xl$ are deontic literals.

In addition we introduce the notion of \otimes -expressions.

- every literal is an \otimes -expression;
- if l_1, \dots, l_n are literals, then $l_1 \otimes \dots \otimes l_n$ is an \otimes -expression

In FCL each business rule statement or any condition applying on a process is represented by a rule, where a rule is an expression

$$r : A_1, \dots, A_n \Longrightarrow_O C$$

where r is the ID or name of a business rule statement, A_1, \dots, A_n is the *antecedent* of rule and C is conclusion of the rule. Each A_i is either a literal or deontic literal and C is an \otimes -expression. The meaning of the above expression is that normative position (e.g., obligations) represented by the conclusion of the rules is in force provided all premises of the rule hold. By using the \otimes connective, we can combine the primary as well as contrary to duty obligations to form a unique rule e.g. $A \otimes B \otimes C$. The meaning of such expressions is very simple: A is the primary obligation, but if A is violated or not done, then B becomes the obligation as a replacement of A . Thus B becomes a reparation of the violation of A which means that A does not hold but the negation of A i.e. $\neg A$ holds. In addition, in case if B also fails, then now it is required to fulfill the obligation of C . Suppose we have the rules

$$r_1 : a \Longrightarrow_O b \qquad r_2 : c, Ob \Longrightarrow_O \neg d \otimes e$$

and we have that a and c and d hold. From the first rule we obtain the b is obligatory (Ob), and then we can apply r_2 . This rule produces $O\neg d$ (d is forbidden or $\neg d$ is obligatory). Rule r_2 also states that the violation of the prohibition of d is compensated by e . Thus, since we have d , we violated the prohibition, and now we have the obligation to compensate it, that is Oe . See [16] for full details of FCL.

5 Modeling Control Objective and Business Rules

Based on the compliance methodology proposed by [2], we generate control objectives corresponding to the business rules given above. For each control objective we identify the relevant document, data items and constraints. We present how FCL intuitively captures the meanings of business rules and provides a faithful representation of normative specifications for lodgment verification process scenario presented in Section 2.

The first step is to introduce the logical predicates needed for the representation of the control objectives and business rules. Here we assume that the relevant data is stored in a database. We will have two types of predicates. The predicates in the first

class essentially correspond to the attributes in the database. Thus we have predicates representing the tables and the attributes in the database. In the second class we have the predicate $contains(x, y)$. The meaning of it is ‘document x contains information/data value y ’.

In the first class we have the predicates (with their meaning)

- $eLC(x)$: x is an Electronic Lodgment Case;
- $eCIR(x)$: x is an eConsent Information Report;
- $eLIR(x)$: x is an eLodgment Information Report;
- $eNoSIR(x)$: x is a Notice of Sale information Report;
- $ELNws(x)$: x is a ELN workspace.

We are now ready to provide the control objectives and the formalization of the rules.

Business Rules: BR1 and BR2

Document Type: eConsent Information Report

Data Item: ELN Workspace ID

Constraints on Data Item:

1. Exactly one must be present
2. Must be the same as the ELN workspace ID in the eLodgment information report in the same elodgment case

Mapping:

$$r_{1,1} : ELNws(x), eCIR(y) \implies_O contains(y, x)$$

$$r_{1,2} : ELNws(x), eCIR(y), ELNws(z), x \neq z, contains(y, x) \implies_O \neg contains(y, z)$$

$$r_2 : ELNws(x), eCIR(y), eLIR(z), eLC(u), contains(y, x), contains(u, y), \\ contains(u, z) \implies_O contains(z, x)$$

The meaning of this formal representation is that, the predicate $ELNws$ must be present exactly once in the *eConsent Information Report* $eCIR$. Rule $r_{1,1}$ specifies that if x is the ID of an ELN workspace, and y is the ID of eConsent Information Report, then it is obligatory that the value of the workspace ID appears in the eConsent Information Report. Rule $r_{1,2}$ states that if x and z are different workspaces (workspace IDs) and one of them is present in the eConsent Information Report identified by y , then it is forbidden for the other workspace ID to appear in y .

Rule r_2 first identifies the type of several documents (e.g. eConsent Information Report, eLodgment Information Report, eLodgment Case), and the ID for the ELN workspace. In addition, if the eConsent Information Report contains a reference to a ELN workspace ID ($contains(y, x)$), and the eConsent Information Report is part of an eLodgment Case ($contains(u, y)$), and there is an eLodgment Information Report that is part of the same application ($contains(u, z)$), then the eLodgment Information Report must contain a reference to the same ELN workspace $contains(z, x)$.

Business Rules: BR3 and BR4

Document Type: eConsent Information Report

Data Item: ELN eLodgment Case ID

Constraints on Data Item:

1. Exactly one must be present
2. Must be the same as the ELN elodgment case ID in the elodgment information report in this elodgment case

Business Rules: BR5 and BR6

Document Type: eNoS Information Report

Data Item:ELN eLodgment Case ID

Constraints on Data Item:

1. Exactly one must be present
2. Must match the ELN elodgment Case ID in the elodgment information report

The formal rules for the control objectives for Business Rules BR3–BR6 have the same formal representation of the rules for BR1 and BR2.

These control objectives and the resulting FCL rules will provide us guidance on (1) what elements of a lodgment document are relevant for compliance, and consequently which tables and attributes must be extracted from the database schema, and (2) how to formally model the business rules.

6 The Schema Extraction: Compliance Methodology

In this section we provide the account of our proposed methodology which explicitly shows how we can extract the schema for the required data from the databases linked to a process. Figure 4 gives an overview of the overall methodology. As was mentioned in Section 1, process models can be enriched with data in the form of data control tags as required by the process to complete a specific task for compliance verification, and the question was raised where these data annotations will come from. We propose to essentially extract these annotations by querying the database created from the analysis of business rules statements. We use abstract business rules with no information on the processes and identified all pertaining entities involved in the data verification process by means of Entity Relationship (ER) diagram as shown in Figure 5. In the data verification process, each lodgment case has several associated documents with defined attributes. These documents may have several identical and distinct attributes, we have not listed all the attributes in the ER diagram and just give a nominal representation of the data items. From the ER model, the database schema for the lodgment case has been extracted comprising several database tables corresponding to each associated document such as *e_consentinformationreport*, *e_registryinstrument*, *enos_informationreport* etc., and tables for request and response messages. The lodgment case table contains information about the lodgment case for which the data verification request message is sent. In the created database, there are a number of system tables that are automatically created containing information about the database such as columns and key constraints tables. Figure 6 shows the schema for lodgment case database consisting of base tables for each of the associated documents and their attributes, data types, and primary keys. We can query the database to extract the predicates (attributes) of our business rules.

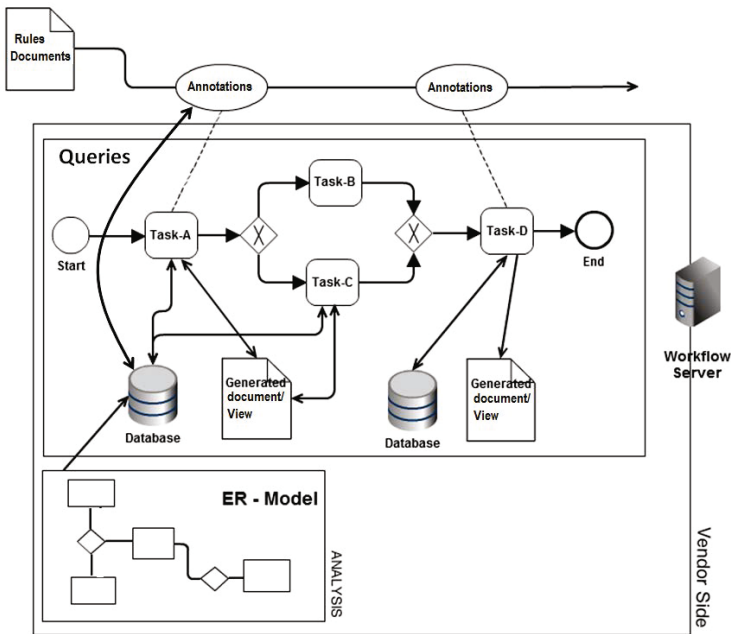


Fig. 4. The Database Schema Extraction Methodology

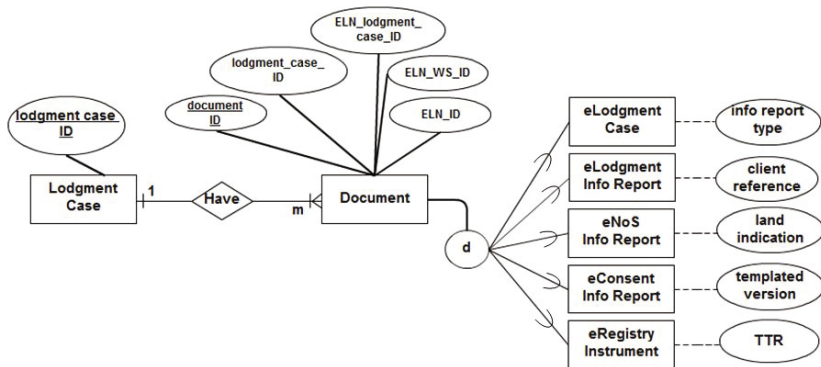


Fig. 5. ER-Diagram for Data Verification

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	DATA_TYPE	COLUMN_KEY
lodgment_case	e_consentinformationreport	elnworkspace_id	varchar	
lodgment_case	e_consentinformationreport	elodgment_caseid	varchar	
lodgment_case	e_consentinformationreport	econsent_informationid	varchar	PRI
lodgment_case	e_consentinformationreport	jurisdiction	tinytext	
lodgment_case	e_consentinformationreport	eln_subscriberid	varchar	
lodgment_case	e_consentinformationreport	e_registryinstrument	varchar	
lodgment_case	e_registryinstrument	registryinstrument_id	varchar	PRI
lodgment_case	e_registryinstrument	elodgment_caseid	varchar	
lodgment_case	e_registryinstrument	jurisdiction	tinytext	
lodgment_case	e_registryinstrument	client_reference	varchar	
lodgment_case	e_registryinstrument	land_registry_delivery_address	varchar	
lodgment_case	e_registryinstrument	elnworkspace_id	varchar	
lodgment_case	elnlodgment_informationreport	informationreport_type	varchar	
lodgment_case	elnlodgment_informationreport	elnlodgmentinformationreport_id	varchar	PRI
lodgment_case	elnlodgment_informationreport	elnworkspace_id	varchar	
lodgment_case	elnlodgment_informationreport	elnenos_id	varchar	
lodgment_case	elnlodgment_informationreport	jurisdiction	text	
lodgment_case	elnlodgment_informationreport	elnsubscriber_id	varchar	
lodgment_case	enos_informationreport	enos_id	varchar	PRI
lodgment_case	enos_informationreport	eln_subscriberid	varchar	

Fig. 6. Column Schema for Lodgment Case Database

The query to extract the predicates *ELNws*, *eCIR*, *eLIR*, *eNoSIR* and *eLC* is²

```
SELECT TABLE_NAME, COLUMN_NAME
FROM SYSTEM_INFORMATION
WHERE COLUMN_KEY = 'PRI'
```

After that we have to take the `TABLE_NAME` as the name of the predicate and the `COLUMN_NAME` as the argument of the predicate. For example, one of such predicates would be *enos_informationreport(enos_id)*, or if we use the mapping with the abbreviations *eNoSIR(enos_id)*. The meaning for ground instances of these predicates is, “column_name” is the primary key of “table”; thus *eNoSIR(enos_id)* means *enos_id* is the identifier of an *eNoSIR* document. This kind of predicates, where the variables (arguments) of the predicates have names, are helpful in other respects. For example it could be used to check conformance (compliance at run time, or that the data in an instance of a process is correct). We can instantiate a predicate using the following schema for a query:

```
SELECT $COLUMN_NAME
FROM $TABLE_NAME
```

² For space and readability reasons, we use abbreviation for the predicates, and full names for the database. It would have been possible to use the same names, or to establish a one-to-one mapping between the elements in the data dictionary of the rules and the elements in the data dictionary of the databases linked to a process.

So to get the extension of the predicate *eNoSIR*, we run the query

```
SELECT ENOS_ID
FROM ENOS_INFORMATIONREPORT
```

The abstract extension of the predicate *contains* can be computed by the following query

```
SELECT X.COLUMN_NAME, Y.COLUMN_NAME
FROM SYSTEM_INFORMATION as X, SYSTEM_INFORMATION as Y
WHERE X.TABLE_NAME = Y.TABLE_NAME
AND X.COLUMN_KEY = 'PRI'
```

The query is a simple self-join of the system information table, using `TABLE_NAME` as the join attribute, and it returns pairs of column names for columns in the same table, where the first is the primary key of the table. Thus for example using the data in Figure 6 we have the pair *registryinstrument_id* and *elodgmentcase_id*. Every row of the table *e_registryinstrument* has the ID of an *e_registryinstrumnt* and the ID of the *elodgmentcase*. In other words each row represents a document of a given type, the information in it, and the primary key represents the document. Thus *registryinstruemnt_id* and *elodgmentcase_id* means that the information report about an *e_registryinstrument* contains a reference to the *elodgmentcase* of which it is part of. Notice that the first query and the last query are domain independent. All we need is a system table with the information on the schema of the database used by a process.

Based on the idea presented above, checking data compliance (of a database schema against a set of business rules defining the data constrains) can be simply performed by running the above query on the database linked to a particular task to get the (data) annotations for that task. After that we can use a two step compliance checking algorithm proposed by [13, 14] which, in the first step, examines each task in the process against all relevant obligations; and generates a status report on active reparation chains. Then, in the second step, it determines if a process is compliant with all regulations or not.

7 Related Work

In recent past a number of approaches focusing on checking compliance on business process models have been reported in literature [3, 20–23]. As we discussed previously about the requirement of a preventive approach *compliance by design* for business process compliance. This literature can be divided into two distinct categories: compliance by design and post design compliance checking. In the first approach new business process models are fed with business rules as input whereas a process model is checked against compliance requirements when a process has completed the design phase.

Lu et. al [24] objectively showed how to enforce compliance requirements to avoid the chance of potential rules violations. Similar works reported by [25–27], although provide good solution to achieve design-time compliance yet compliance checking will be required if changes are made to the process model, and new business rules are introduced. In addition to that, the emphasis of these approaches remained on the structural compliance of a process model, and the data aspect has largely been ignored. Goedertier

and Vantienen [25] achieved design-time business process compliance using rule sets with permissions and obligations and proposed PENELOPE, a declarative language to specify compliance rules. PENELOPE generates a state space and a BPMN model from these rules which is compliant by design. This approach concentrates on acyclic processes only, and the data and data constraints aspect in the business rules is not present. An artifact-centric business process modeling approach has been recently proposed in [28], exhibiting how artifact-centric business processes can be canonically extended to take also compliance rules into account. As these business rules can express constraints on the execution of actions, it is claimed that the data information can also be taken into account but it is not clear whether the model will be semantically annotated with the data, and how data constraints will be modeled. If in case business process model is semantically annotated then where this data will come from.

In the post process model design compliance checking Awad et.al. [29] discussed a temporal logic query based approach for specification, verification and explanation of violations of data-aware compliance rules. The approach employs extended BPMN-Q to realize the business rules including the data aspects to increase the expressiveness of their previously proposed language in [30]. As the authors used *past linear temporal logic* (PLTL) to formalize the business rule, the problem with temporal logic is that it provides structural compliance only, and does not distinguish different normative positions. There is no indication how these normative positions and data associated with these normative positions can be represented. Moreover, this proposed approach comes under the post design compliance checking. To measure the compliance distance between the process model and a rule an automated approach was introduced in [31]. The degree of compliance is checked on a scale from 0 to 1 but the data aspect has not been covered.

Our work reported in this paper falls in latter category to achieve compliance by design. To our end, we believe until recently no work has been reported that specifically extract data schema from the business rules to semantically annotate a process model for compliance checking purpose.

8 Conclusions and Future Work

In this paper we proposed a methodology to automatically extract annotations related to the data schema, and templates linked to the tasks in a process. This exhibits how we can extract the data schema from the database generated from the business rules, including the primary keys of the associated documents of the lodgment verification process presented in Section 2. We see the contribution of this work in different ways: *first* our methodology will provide a better understanding of data annotations from schema related to tasks a the process. *Second*: the BPM process model, at hand, can be extended with the extracted annotations in the form of data tags as proposed in [2]. In addition to that, the extracted schema will also help to model data constraints on the process model for better compliance checking. *Third*: this methodology provides an answer to the question where do we get the data annotations from, if we want to extend a process model with these annotations, and model constraints on a process model.

Currently we have used abstract data, and a hypothetically created process example to present our methodology to show how we can extract annotations from abstract data

to extend a business process model. As this is a preliminary work and has not been implemented yet, we are not aware of any complexity that might arise when extending a process model. On the same note, we are also unable to report, at this stage, what will be the behavior of process model populated with the extracted data schema. Hence we believe this proposed preliminary work requires a large scale industry evaluation and validation for further insights and generalization. In addition to that, due to the varying nature of business process tasks, amount of data used, and data redundancies are prevalent in the business rules statements, extracting a normalized data schema will certainly be a challenge. In our example case scenario, we experienced many redundant data items appearing several times in the business rules, we just used this redundant data to extract preliminary schema. The positive aspect of using FCL is that, it provides normalization functionalities to remove any redundancies in the business rules statements, we believe that this matter is solvable and of further interest. Lastly, but no least, as business rules tend to change frequently, a business analyst can come up with new predicates. This requires further understanding how these changes can be accommodated in the existing database.

Acknowledgements. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Center of Excellence program.

References

1. Australia AIGD. The Australian Industry Group National CEO Survey: Business Regulations (September 2011)
2. Sadiq, S., Governatori, G., Namiri, K.: Modeling Control Objectives for Business Process Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007)
3. Liu, Y., Müller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. IBM Systems Journal 46, 335–361 (2007)
4. Schmidt, R., Bartsch, C., Oberhauser, R.: Ontology-based Representation of Compliance Requirements for Service Processes. In: SBPM 2007. CEUR, vol. 251, pp. 28–39 (2007)
5. El Kharbili, M., Stein, S., Markovic, I., Pulvermüller, E.: Towards a Framework for Semantic Business Process Compliance Management. In: GRCIS 2008 Workshop at CAiSE 2008. CEUR, vol. 339, pp. 1–15 (2007)
6. Hagerty, J.: Sox spending for 2006. ARM Research, Boston, USA (November 2006), JH 2006
7. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., Dadam, P.: SeaFlows Toolset – Compliance Verification Made Easy for Process-Aware Information Systems. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 76–91. Springer, Heidelberg (2011)
8. Ly, L.T., Rinderle, S., Dadam, P.: Semantic Correctness in Adaptive Process Management Systems. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 193–208. Springer, Heidelberg (2006)
9. Governatori, G., Sadiq, S.: The Journey to Business Process Compliance. In: Handbook of Research on Business Process Management, pp. 426–454. IGI Global (2009)
10. Sadiq, S., Governatori, G.: A Methodological Framework for Aligning Business Processes and Regulatory Compliance. In: Handbook of Business Process Management: 2. Strategic Alignment, Governance, People and Culture, pp. 159–176. Springer, Berlin (2010)

11. Ghose, A.K., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
12. Governatori, G., Hoffmann, J., Sadiq, S., Weber, I.: Detecting Regulatory Compliance for Business Process Models through Semantic Annotations. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM Workshops 2008. LNBP, vol. 17, pp. 5–17. Springer, Heidelberg (2009)
13. Governatori, G., Rotolo, A.: An Algorithm for Business Process Compliance. In: Legal Knowledge and Information Systems, pp. 186–191. IOS Press (2008)
14. Governatori, G., Rotolo, A.: A Conceptually Rich Model of Business Process Compliance. In: APCCM 2010. CRPIT, vol. 110, pp. 3–12. Australian Computer Society (2010)
15. Carmo, J., Jones, J.: Deontic Logic and Contrary to duties. In: Handbook of Philosophical Logic, 2nd edn., pp. 265–343. Kulwer, Dordrech (2002)
16. Governatori, G.: Representing Business Contracts in RuleML. *International Journal of Cooperative Information Systems* 14, 181–216 (2005)
17. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. *ACM Transactions on Computational Logic* 2, 255–287 (2001)
18. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming* 6, 703–735 (2006)
19. Governatori, G., Rotolo, A.: Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligation. *Australasian Journal of Logic* 4, 193–215 (2006)
20. Bonazzi, R., Pigneur, Y.: Compliance Management in Multi-actor Contexts. In: GRCIS 2009. CEUR, vol. 459, article 7 (2009)
21. COMPAS: Compliance Driven Models, Languages, and Architectures for Services. In: 7th Framework Programme Information and Communication Technologies (2008)
22. el Kharbili, M., Stein, S.: Policy-Based Semantic Compliance Checking for Business Process Management. In: MobIS Workshops 2008. CEUR, vol. 420, pp. 178–192 (2008)
23. Ly, L., Rinderle-Ma, S., Göser, K., Dadam, P.: On Enabling Integrated Process Compliance with Semantic Constraints in Process Management Systems. *Information Systems Frontiers* 14, 195–219 (2012)
24. Lu, R., Sadiq, S.K., Governatori, G.: Compliance Aware Business Process Design. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer, Heidelberg (2008)
25. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 5–14. Springer, Heidelberg (2006)
26. Governatori, G., Milosevic, Z., Sadiq, S.: Translating business contract into compliant business processes. In: 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 221–232. IEEE Press (2006)
27. Goedertier, S., Vanthienen, J.: Compliant and flexible business processes with business rules. In: BPMDS 2006. CEUR, vol. 236 (2006)
28. Lohmann, N.: Compliance by Design for Artifact-Centric Business Processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 99–115. Springer, Heidelberg (2011)
29. Awad, A., Weidlich, M., Weske, M.: Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 500–515. Springer, Heidelberg (2009)
30. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
31. Lu, R., Sadiq, S., Governatori, G.: Measurement of compliance distance in business processes. *Information Systems Management* 25, 344–355 (2008)

Formalizing Both Refraction-Based and Sequential Executions of Production Rule Programs

Bruno Berstel-Da Silva

Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Germany

Abstract. Production systems are declarative, in that they do not explicitly specify the control flow. Yet, the concept of a production system does not include the definition of a given control strategy. The control between rules in a production rule program is, in practice, defined by each implementation of a production rule engine. Engines have traditionally been implemented using the Rete algorithm. Since the turn of the century, however, production systems have evolved into industrial products known as Business Rules Management Systems (BRMS). BRMS have introduced new compilation and execution schemes, which are often called *sequential* in contrast with the incremental behavior of Rete. This change in execution scheme came with a change in semantics for rule programs. In this paper, we propose a formal description of the execution of production rule programs. Existing descriptions either ignore the control strategy, or assume a Rete semantics. Ours isolates the handling of rule eligibility in the control strategy, which allows us to describe the sequential execution semantics of rule programs, as well as the Rete semantics, and others.

1 Introduction

Business Rules Management Systems (BRMS) [7,11,16,21,22,26] are industrial products that have gained substantial consideration as a way to lower the cost of frequent changes in business policies [18,29]. The contribution of BRMS is to externalize the business logic of an application as a rule program, and to provide business experts with tools to author and manage these rules collaboratively.

The rules in question are of the condition-action type, also known as production rules. As such, BRMS can be seen as descendants of production systems [12,19], in the tradition of OPS5 [3,13]. Indeed, the rule engines present in most BRMS compile and execute rule programs with variants of the Rete algorithm [14,15,20], which is at the heart of OPS5. However, with the transition from production systems to BRMS, rule programs have evolved from medium-sized programs implementing inference algorithms to massive programs performing simpler tasks [28]. In this context, the concern for the implementors of rule engines has shifted from providing smart control strategies in the execution

of programs to designing new compilation schemes that would ensure a high throughput in the processing of data.

This new generation of compilation and execution algorithms is often referred to as *sequential* [10,17,23], both due to the way they consume input data and in contrast with the incremental behavior of Rete. Yet, the gain in performance has been achieved at the price of a change in the execution semantics of the rule programs. The main difference between the Rete and sequential semantics is that the former allows rule programs to implement complex inference schemes, whereas the latter makes assumptions (a stable working memory, a static rule set, etc.) that opens the door to faster execution. This disruption in semantics is usually acceptable for the users of BRMS, because their rule programs, although important in size, implement simple algorithms that do not require evolved inference mechanisms.

This paper is organized as follows. In Sect. 2, we expose our formal description of rule program execution. Then, in Sects. 3 and 4, we review some selection and eligibility strategies; in particular, we provide the formalization of Rete's refraction semantics and of the sequential semantics of modern BRMS. Finally, in Sect. 5 we illustrate both semantics and their formalizations on an example rule-based application, which we introduce in Sect. 1.2.

1.1 Related Work

Up to now, and to the best of our knowledge, there have been only few formalizations of the execution behavior of production rule programs with the Rete semantics, and none with the sequential semantics. Such a formalization is however useful to develop tools that help understand, verify, test, and more generally analyze, rule programs. In this paper, we present a logic-based description of rule programs and of their execution, with a focus on the distinction between the applicability and the eligibility of a rule.

In the traditional presentations of Rete, eligibility is one step of the control strategy, called *refraction*. By isolating the eligibility strategy, our formalization allows us to depict the execution of rule programs with either Rete's refraction semantics, the sequential semantics, or others. It is based on first-order logic; we also use objects with attributes to reflect the fact that rule programs in BRMS handle object-oriented data provided by an embedding application. We model states as first-order logic structures and rule execution as relations between states.

Production systems have been formalized in a number of ways, either based on first-order logic or not. In [27] and in previous publications, Schmolze and Snyder have explored the connection between production rules and term rewriting systems. However, their approach focuses on confluence and termination properties, and does not aim at describing the execution behavior of production systems.

Production systems have also been studied from the viewpoint of active databases (see [1] for a survey). Again, confluence and termination is the main focus, and the composition of rules into a rule program execution is not addressed. Safety properties are treated in [2], and mapped to constraint

satisfiability problems on the transition constraints that describe the execution of the rule program. However, the construction of these constraints is not studied. In [4], propositional production systems are modeled with μ -calculus, and production systems with variables are modeled with fixed-point logic. This approach leads to using first-order logic structures as we do.

The formalizations of production systems that address their execution behavior are due to Fages and Lissajoux [9], to Cirstea *et al.* [6], and to Damásio *et al.* [8,25]. These formalizations are based on first-order logic, with [8] relying on Answer-Set Programming. All consider the execution of rule programs by the Rete algorithm. However, [9] explicitly discards the control strategy from its scope and provides a nondeterministic view of rule program execution. On the other hand, [6] includes the control strategy in its formal description of rule program execution, but does not go into any detail; furthermore, the examples given implement an explicit control flow and hence do not exhibit the role of the control strategy. Finally, [8] does include the control strategy in its formalization of rule programs semantics.

1.2 Running Example

Acme.com is an e-commerce company. It wants to introduce a rewarding program in which customers earn bonus points, and chooses to implement it with a BRMS. Customers are entered into the system with their current bonuses and their purchases, to be processed by the rule program $\mathcal{R} = \{P, S\}$ that implements the rewarding program.

The first rule in the program grants a customer bonus points, based on his/her purchases. Namely, if the value of the purchase p exceeds \$ 100, the customer c earns 10% of the purchase value in bonus points.

$$P(c, p): c = p.\text{buyer} \ \& \ p.\text{value} \geq 100 \rightarrow c.\text{bonus} := c.\text{bonus} + p.\text{value} \times .1$$

The second rule implements a sponsorship mechanism: if a customer s sponsors another customer c , then a transfer of bonus points occurs; the transfer only occurs if the sponsor has at least 200 points.

$$S(s, c): s = c.\text{sponsor} \ \& \ s.\text{bonus} \geq 200 \rightarrow \begin{array}{l} s.\text{bonus} := s.\text{bonus} - 50; \\ c.\text{bonus} := c.\text{bonus} + 30 \end{array}$$

This short example introduces rules, with a guard that determines when the rule applies, and an action that indicates how the rule execution will evolve the system state by performing updates on object attributes. The attributes in this example program are *bonus*, *value*, *buyer*, and *sponsor*. Objects are held by rule variables: the rule variables in P are c and p , those in S are s and c . These concepts are formalized in Sect. 2.

In contrast with industrial rule languages or with RIF-PRD [25], our language does not include adding objects to, or removing objects from, the working memory. However, like industrial rule languages, it regards the update of an object attribute as a single operation.

2 Formal Description of Rule Program Execution

2.1 Expressions, States, Rules, Programs

A rule program implements part of the logic of a larger application. To this end, the rules handle the data of the application, in the format defined by the application itself. In practice, they use the data types of the embedding language, such as Booleans, numbers, enumerations, and, in an object-oriented context, the classes defined by the application. From a formal viewpoint, the application introduces a *theory* Θ , which contributes to the *signature* of the rule language with a set of function symbols (including constants), and provides their interpretation. Classically, the theory Θ can include Booleans with logical connectors (\wedge , \vee , \neg , ...), numbers with arithmetic operators ($+$, $-$, \times , ...), uninterpreted functions with equality, etc. In our framework, we assume that Θ includes at least Booleans, without quantifiers, and objects with attributes as defined below. The running example exposed in Sect. [L.2](#) also includes numbers.

Objects have a unique identity. The identities of two objects can be compared for equality; no other operation is available on the identity of objects. Object *attribute symbols* are unary function symbols.

In our rule language, objects are handled through *variables*. *Expressions* are built in the classical, inductive way on the signature inherited from Θ . They include arithmetic and Boolean expressions, but also *attribute references*. An attribute reference denotes the value of an object's attribute; it is written in dotted postfix notation. For example, *p.age* refers to the attribute with symbol *age* of the object held by the variable *p*.

A *state* is a first-order logic structure. The common domain to all states is provided by the theory Θ ; we note it \mathcal{D} . As mentioned previously, it includes at least the Booleans and the objects, the infinite set of which we note \mathbb{O} . The interpretation function \mathcal{I}_s of a state *s* interprets function symbols as specified by the theory Θ —typically, logical connectors and arithmetic operators are interpreted in the classical way. Attribute symbols are interpreted by partial functions from \mathbb{O} to \mathcal{D} . For an attribute symbol *f*, we note $\mathcal{I}_s(f)$ or f^s the function that interprets *f* in *s*. The definition domain of this partial function is noted $\text{Dom}(f^s)$.

A *rule* $r = (\vec{o}, g, a)$ consists of the tuple $\vec{o} = (o_1, \dots, o_m)$ of its *rule variables*, the Boolean expression *g* called its *guard*, and its *action* *a*, described further below. The *arity* of *r*, noted $|r|$, is *m*. A *rule instance* is a tuple $R = (r, O_1, \dots, O_m)$ where $O_1, \dots, O_m \in \mathbb{O}$ are objects. The objects in a rule instance provide values for the rule variables, which are used to interpret the rule guard and action, as described below.

A rule instance $R = (r, O_1, \dots, O_m)$ is *applicable* in a state *s* if the guard *g* of *r* holds in this state and on the objects O_1, \dots, O_m . That is, if the interpretation of the Boolean expression *g* by state *s*, with each variable o_j mapped onto the object O_j for $j = 1, \dots, m$, yields true. The guard *g* is interpreted as false by *s* if one has $O_j \notin \text{Dom}(f^s)$ for any attribute reference $o_j.f$ that appears in *g*.

When a rule instance $R = (r, O_1, \dots, O_m)$ is **applied**, the action a of the rule is executed on the objects in the instance. A rule action is a sequence of assignments to attributes of rule variables. An *assignment* is a statement $o_{j_k}.f_k := e_k$ that denotes the update of the attribute f_k for the object held by o_{j_k} with the value of the expression e_k . Its semantics, when executed from a state s , is to produce a new state s' , in which all attribute and function symbols are interpreted as in s , with the exception of f_k . In state s' , the attribute symbol f_k is interpreted by the partial function $f_k^{s'}$ with the same domain as f_k^s , and such that

$$\forall O \in \text{Dom}(f_k^s) \quad f_k^{s'}(O) = \begin{cases} \mathcal{I}_{s'}(e[O_j/o_j]_{j=1}^m) & \text{if } O = O_{j_k} \\ f_k^s(O) & \text{otherwise.} \end{cases}$$

To summarize, given a rule $r = (\vec{o}, g, a)$, a rule instance $R = (r, O_1, \dots, O_m)$ is applicable in a state s if the guard g holds in s and on the objects O_1, \dots, O_m . When R is applicable in s , then the application of R in s produces a new state s' that results from the execution of the action a from s on O_1, \dots, O_m . We note this application $s \xrightarrow{R} s'$.

A **rule program** $\mathcal{R} = \{r_1, \dots, r_n\}$ is a finite set of rules. A rule program is executed on a finite set of objects $\mathcal{M} \subset \mathbb{O}$, called the *working memory*. The set of all rule instances that can be formed out of rules in \mathcal{R} and objects in \mathcal{M} is noted $\mathcal{I}(\mathcal{R}, \mathcal{M}) = \{(r, \vec{O}) \mid r \in \mathcal{R}, \vec{O} \in \mathbb{O}^{|\mathcal{R}|}\}$. Given a state s , the subset of rule instances that are applicable in s is noted A_s .

2.2 Execution of a Rule Program

To formally describe the execution of a rule program, we introduce the notion of a *configuration* of the rule engine, as a pair $\langle E, s \rangle$, where $E \subseteq \mathcal{I}(\mathcal{R}, \mathcal{M})$ is a set of rule instances and s is a state. In such a configuration, E denotes the set of *eligible* rule instances in s .

With this definition, we say that an *execution of a rule program* \mathcal{R} on a working memory \mathcal{M} from an initial state s_0 is defined by a potentially infinite sequence of transitions between configurations

$$\langle \mathcal{I}(\mathcal{R}, \mathcal{M}), s_0 \rangle \xrightarrow{R_1} \langle E_1, s_1 \rangle \xrightarrow{R_2} \langle E_2, s_2 \rangle \xrightarrow{R_3} \dots$$

in which each transition complies with the following transition rule, for $k > 0$:

$$\frac{\{R_k\} = \mathcal{S}(A_{s_{k-1}} \cap E_{k-1}) \quad s_{k-1} \xrightarrow{R_k} s_k \quad E_k = \mathcal{E}(E_{k-1}, \dots)}{\langle E_{k-1}, s_{k-1} \rangle \xrightarrow{R_k} \langle E_k, s_k \rangle}. \quad (1)$$

This transition rule encodes that the rule engine can perform a transition by rule instance R_k from configuration $\langle E_{k-1}, s_{k-1} \rangle$ to configuration $\langle E_k, s_k \rangle$ if the following conditions are met: the rule instance R_k is selected among the rule instances that are both applicable and eligible in s_{k-1} , and its application in s_{k-1} produces the state s_k .

This transition rule exhibits the selection strategy \mathcal{S} and the eligibility strategy \mathcal{E} as two parameters of the rule engine semantics. The selection strategy \mathcal{S} is a function that takes a set of rule instances and returns either the empty set or a singleton included in the set received. Sect. 3 reviews the most common selection strategies.

Which rule instances are eligible at each step of the execution is determined by the eligibility strategy \mathcal{E} , based on the set of previously eligible rule instances and on possibly other arguments, such as the rule instance being applied, the initial or final states of the transition, etc. At the beginning of the execution, all rule instances are eligible. Sect. 4 reviews the eligibility strategies at work in the Rete and sequential execution semantics.

As an extreme case, consider the eligibility strategy \mathcal{E}_{id} that always returns the previous set of eligible rule instances, and the selection strategy \mathcal{S}_{nd} that non-deterministically selects a rule instance. These strategies give the rule program execution semantics considered by [9].

2.3 Comparison with Traditional Presentations

The control strategy in a production rule engine is analogous to the scheduler in parallel programming or to the method call resolution in an object-oriented language. In all these control mechanisms, a variation on any criterion can change the course of program execution dramatically. However, in contrast with object-oriented languages where the method call semantics is defined with the language, rule languages do not include the definition of a control strategy. Instead, the control strategy is brought in by the algorithm used to execute the rule program.

Since the seminal implementation by OPS5, production systems have traditionally executed rule programs with the Rete algorithm or a variant. More or less formal descriptions of its control strategy can be found in the OPS5 User's Manual [13], in the RIF-PRD recommendation [25], in papers describing extensions to OPS5 [5,24], or in the documentations of BRMS [17].

The Rete control strategy applies to the set A_s of applicable rule instances in the current state s . In the context of Rete, this set is called the *conflict set* [3,19]. It is traditionally presented as the following four steps. If, after the first step, the conflict set is empty, then the program execution stops. If, as the result of any step, the conflict set contains only one rule instance, then this rule instance is selected for application.

- (i) *Refraction*: Discard from the conflict set any rule instance that has already been applied, and has since remained applicable.
- (ii) *Recency*: Retain the rule instances that include objects that have been inserted or modified last.
- (iii) *Specificity*: Retain the rule instances that relate to the most specific rules. (We define rule specificity in Sect. 3.)
- (iv) *Random*: Arbitrarily retain only one rule instance.

Implementations of production systems other than OPS5 [7,11,16,21,22,26] have adopted similar control strategies, with minor variations on the definition of refraction, and a wider range of criteria to select the rule instance to apply among those that successfully passed the refraction step.

From a broader perspective, one can identify two purposes in a control strategy for the execution of rule programs. A first goal is to discard program executions that do not make sense, for example to avoid trivial loops. In the Rete algorithm, this is the role of the refraction step. Our formalism exposed in Sect. 2.2 generalizes this filtering task with the eligibility strategy. A second goal is to choose one rule instance among the ones that have passed the filtering step. This is the role of the three last steps of Rete. We generalize this task with the selection strategy. The distinction that our framework introduces between the selection and eligibility strategies allows us to describe other execution schemes than Rete, as shown in Sect. 4.

3 Selection Strategies

When several rule instances are both applicable and eligible in a given state, the choice of which to apply in the transition to the next configuration is the role of the selection strategy. In the traditional Rete control strategy, exposed in Sect. 2.3, this is addressed by the last three steps.

In practice, selection strategies define an order on rule instances, and return the (applicable and eligible) instance that is maximal according to this order. The order is classically defined by the lexicographic combination of various orders such as the following ones [3,19].

- *Priority on rule instances.* A rule $r = (\vec{o}, g, a)$ is equipped with a numerical expression π_r in the rule variables. The order is based on the value of this expression for each rule instance in the current state.
- *Priority on rules.* This order is a simplified version of the previous one, where the expressions π_r are numerical constants. The priority of all instances of a rule r are then equal to the number π_r , independently of the state.
- *Strict ordering of the rules.* A strict order is explicitly defined on the rules, for example by setting the rule priorities to a permutation of $\{1, \dots, n\}$.
- *Specificity of the rules.* A rule $r_1 = (\vec{o}_1, g_1, a_1)$ is said to be more specific than another rule $r_2 = (\vec{o}_2, g_2, a_2)$ when one has $g_1 \Rightarrow g_2$. This defines a partial order on rules. Some rule engines approximate this order by using empirical indications of the rule specificity, such as the number of elementary Boolean expressions in the rule guard, or the arity of the rule.
- *Recency.* This order is based on a numerical constant associated with each object in the working memory, called the object recency, with the idea that objects have been inserted into the working memory in some order. The recency of a rule instance is given by the maximal recency of the objects in the rule instance.

4 Eligibility Strategies

4.1 The Refraction Eligibility Strategy

As stated in the RIF-PRD recommendation [25]: “The essential idea of refraction is that a given instance of a rule must not be fired more than once as long as the reasons that made it eligible for firing hold.” As a direct consequence, enforcing refraction must take the execution history into account. To this end, [25] chooses to define refraction by counting during how many execution steps each rule instance has remained applicable, and since how many steps it has been applied.

In our framework, we base the definition of refraction on the set E_k of eligible rule instances in a configuration $\langle E_k, s_k \rangle$ of the rule engine. Using the formalism introduced in Sect. 2.2, refraction can be defined as follows.

Refraction. If a rule instance R has been applied in a configuration transition $\langle E_i, s_i \rangle \xrightarrow{R} \langle E_{i+1}, s_{i+1} \rangle$, it is eligible for application in a subsequent transition $\langle E_k, s_k \rangle \xrightarrow{R} \langle E_{k+1}, s_{k+1} \rangle$ only if the execution contains a configuration $\langle E_j, s_j \rangle$ such that $i + 1 \leq j \leq k$ and $R \notin A_{s_j}$.

Of course, for the transition $\langle E_k, s_k \rangle \xrightarrow{R} \langle E_{k+1}, s_{k+1} \rangle$ to occur, the rule instance R will have to be applicable in s_k , which implies $j < k$. However, this requirement does not relate to eligibility. The independence between applicability and eligibility is visible in the example discussed in Sect. 5. For instance, we shall see that in state σ_1 , the rule instance (S, Alice, Bob) is eligible but not applicable; and that in σ_3 , it is applicable but not eligible.

The task of the eligibility strategy is to compute the set of eligible rule instances that results from a transition between configurations of the rule engine. As per the definition of refraction above, the refraction eligibility strategy \mathcal{E}_{ref} makes a rule instance ineligible when it is applied, and makes it eligible again as soon as it becomes inapplicable. That is, in a transition $\langle E, s \rangle \xrightarrow{R} \langle E', s' \rangle$:

$$\mathcal{E}_{\text{ref}}(E, R, s') = E \setminus \{R\} \cup \{R' \in \mathcal{I}(\mathcal{R}, \mathcal{M}) \mid R' \text{ is not applicable in } s'\}.$$

In this definition, the eligibility strategy first removes the rule instance R that was just applied from the set of eligible rule instances, which corresponds to the statement “a rule instance must not be fired more than once”. Then, the strategy implements “as long as the reasons that made it eligible for firing hold” by adding the rule instances that became inapplicable due the application of R .

Note that, although the definition of \mathcal{E}_{ref} refers to any rule instance inapplicable in s' , the rule instances added are precisely those that have been already applied in this rule program execution and that have been made inapplicable by the application of R . Indeed, the rule instances that have never been applied are in E since the beginning of the rule program execution; and the rule instances that were already inapplicable in s have been added to E in a previous transition.

Since, in the execution of the rule program, the selection is performed on $A_s \cap E$, the definition of the refraction eligibility strategy above ensures that once applied, a rule instance will not be applied a second time before it becomes first inapplicable and then applicable again, as stated by the Rete algorithm.

4.2 The Sequential Control Strategy

Definition of Sequential Execution. Sequential execution schemes have appeared in BRMS about a decade ago [10,17,23], as an answer to the evolution of rule programs from the many patterns/many objects case for which Rete had been invented, to a many rules/few objects case found in business applications. In this section, we describe the sequential execution mode of IBM Websphere Operational Decision Management [16]; the corresponding execution algorithms in other BRMS [7,11,21,22,26] have similar behaviors.

The sequential execution mode considers all object tuples from the working memory in sequence, and submits each tuple to all rules, in sequence again. If the guard of a rule holds on an object tuple when they are considered together, the rule action is executed. Otherwise, the next rule is considered. There is no “second chance:” a rule instance that has already been considered and would become applicable only later, due to the application of another rule instance, will not be applied. The orders defined on object tuples and on rules are therefore crucial. Any criterion described in Sect. 3 can be chosen, although simple rule priorities are commonly used.

The semantics of the sequential execution mode can be considered poorer, since it suppresses opportunities for inference and chaining between rules. However, it is considered an acceptable trade-off by BRMS users as it allows faster execution. Furthermore, the greater control it provides through the explicit ordering of rules is found opportune with large rule programs.

The Sequential Control Strategy. Unlike refraction, which is a pure eligibility strategy and hence can be combined with any selection strategy, the sequential control strategy defines both a selection and an eligibility strategy. These two strategies \mathcal{S}_{seq} and \mathcal{E}_{seq} implement the behavior just exposed. Namely, the eligibility strategy defines the set of rule instances under consideration, and the selection strategy ensures that the rule instances are picked in the proper order.

As described above, the sequential control strategy is based on a strict ordering of the rules in the rule program, and of the objects in the working memory. These two orderings define a strict order $<_{\text{seq}}$ on rule instances lexicographically. The sequential control strategy is then defined as follows:

- (i) The *selection strategy* \mathcal{S}_{seq} returns the minimal rule instance under consideration:

$$\mathcal{S}_{\text{seq}}(C) = \{R \in C \mid R \text{ is minimal with respect to } <_{\text{seq}}\}.$$

- (ii) The *eligibility strategy* \mathcal{E}_{seq} retains for further consideration only the rule instances that follow (are greater than) the rule instance just applied; in a transition $\langle E, s \rangle \xrightarrow{R} \langle E', s' \rangle$:

$$\mathcal{E}_{\text{seq}}(E, R) = \{R' \in E \mid R <_{\text{seq}} R'\}.$$

5 Illustration

Let us consider again the example rule-based application depicted in Sect. 1.2. The rules of the program $\mathcal{R} = \{P, S\}$ are reminded below:

$$\begin{aligned} P(c, p) : c = p.buyer \ \& \ p.value \geq 100 \ \rightarrow \ c.bonus := c.bonus + p.value \times .1 \\ S(s, c) : s = c.sponsor \ \& \ s.bonus \geq 200 \ \rightarrow \ s.bonus := s.bonus - 50 ; \\ & \ c.bonus := c.bonus + 30 \end{aligned}$$

As seen in Sect. 2, a rule program such as \mathcal{R} is executed on a finite set of objects, called the working memory. Assume that we plan to execute \mathcal{R} on a working memory $\mathcal{M} = \{\text{Alice}, \text{Bob}, \text{Don}, \text{Car}\}$ containing four objects: three customers and a purchase. An execution of \mathcal{R} on \mathcal{M} from an initial state s_0 consists of a sequence of transitions between configurations, starting from $\langle \mathcal{I}(\mathcal{R}, \mathcal{M}), s_0 \rangle$. Assume that the execution starts from a state s_0 in which Alice is a sponsor of Bob and Don, their respective bonuses are 230, 100, and 50 points, and Alice purchases a car for \$900. State s_0 can be depicted as follows:

$$s_0 \left\{ \begin{array}{ll} \text{Alice} : & \text{bonus} = 230 \\ \text{Bob} : & \text{bonus} = 100 \quad \text{sponsor} = \text{Alice} \\ \text{Car} : & \text{buyer} = \text{Alice} \quad \text{value} = 900 \\ \text{Don} : & \text{bonus} = 50 \quad \text{sponsor} = \text{Alice} . \end{array} \right.$$

5.1 Sequential Executions of the Rule Program

The sequential executions of \mathcal{R} will be governed by two orders: the order on object tuples, here customer pairs or customer-purchase pairs; and the order on rules, here P and S. In our example, the rules apply to distinct types of objects; as a result, their order has no impact. Let us assume that S comes before P. On the other hand, the order on object tuples determines which of earning bonus points on purchases, or of sponsoring friends, is favored.

Sponsorship over Purchases. Let us first assume that pairs of customers come before customer-purchase pairs, and that object tuples are taken in the following order:

$$(\text{Alice}, \text{Alice}) (\text{Alice}, \text{Bob}) \dots (\text{Don}, \text{Don}) (\text{Alice}, \text{Car}) (\text{Bob}, \text{Car}) (\text{Don}, \text{Car}) .$$

By definition, all rule instances are eligible in the initial configuration. The sequential selection strategy will hence pick the first applicable rule instance that can be formed for rules P and S, with each of the object tuples in the order above.

Per the definitions of Sect. 2.1, the rule instance $(P, \text{Alice}, \text{Alice})$ is not applicable in s_0 , because $\text{Alice} \notin \text{Dom}(\text{buyer}^{s_0})$. On the other hand, $(S, \text{Alice}, \text{Alice})$ is not applicable in s_0 either, because $\text{sponsor}^{s_0}(\text{Alice}) \neq \text{Alice}$. Similarly, $(P, \text{Alice}, \text{Bob})$ is not applicable in s_0 . However, $(S, \text{Alice}, \text{Bob})$ is applicable in s_0 , and is thus selected by the sequential selection strategy.

Therefore, a transition by $(S, \text{Alice}, \text{Bob})$ from $\langle \mathcal{I}(\mathcal{R}, \mathcal{M}), s_0 \rangle$ to $\langle E_1, s_1 \rangle$ occurs, where the state s_1 results from the application of $(S, \text{Alice}, \text{Bob})$ in s_0 , that is:

$$s_1 \left\{ \begin{array}{ll} \text{Alice:} & \text{bonus} = 180 \\ \text{Bob :} & \text{bonus} = 130 \quad \text{sponsor} = \text{Alice} \\ \text{Car :} & \text{buyer} = \text{Alice} \quad \text{value} = 900 \\ \text{Don :} & \text{bonus} = 50 \quad \text{sponsor} = \text{Alice} \end{array} \right.$$

and the set E_1 contains the rule instances that are greater than $(S, \text{Alice}, \text{Bob})$, that is:

$$E_1 = \{(P, \text{Alice}, \text{Don}), (S, \text{Alice}, \text{Don}), (P, \text{Bob}, \text{Alice}), \dots, (S, \text{Don}, \text{Car})\}.$$

Because Alice has less than 200 bonus points in state s_1 , the first applicable rule instance in s_1 among those in E_1 is $(P, \text{Alice}, \text{Car})$. A transition by this rule instance therefore occurs to $\langle E_2, s_2 \rangle$, where the state s_2 results from the application of $(P, \text{Alice}, \text{Car})$ in s_1 , that is:

$$s_2 \left\{ \begin{array}{ll} \text{Alice:} & \text{bonus} = 270 \\ \text{Bob :} & \text{bonus} = 130 \quad \text{sponsor} = \text{Alice} \\ \text{Car :} & \text{buyer} = \text{Alice} \quad \text{value} = 900 \\ \text{Don :} & \text{bonus} = 50 \quad \text{sponsor} = \text{Alice} \end{array} \right.$$

and the set E_2 contains the rule instances that are greater than $(P, \text{Alice}, \text{Car})$, that is:

$$E_2 = \{(S, \text{Alice}, \text{Car}), (P, \text{Bob}, \text{Car}), (S, \text{Bob}, \text{Car}), (P, \text{Don}, \text{Car}), (S, \text{Don}, \text{Car})\}.$$

None of these rules is applicable in s_2 , that is, $A_{s_2} \cap E_2 = \emptyset$. Therefore, the transition rule **(II)** cannot be applied and the execution of \mathcal{R} ends.

One can note that $(S, \text{Alice}, \text{Don})$ was not applicable when it was considered in s_1 , but is now applicable in s_2 . However, the sequential nature of the execution, enforced by the sequential eligibility strategy, causes it to not be included in E_2 , and hence not to be considered for execution in s_2 .

Purchases over Sponsorship. Let us now assume that customer-purchase pairs come before customer pairs, and that object tuples are thus taken in the following order:

$$(\text{Alice}, \text{Car}) (\text{Bob}, \text{Car}) (\text{Don}, \text{Car}) (\text{Alice}, \text{Alice}) (\text{Alice}, \text{Bob}) \dots (\text{Don}, \text{Don}).$$

With this order on object tuples, the execution of \mathcal{R} on \mathcal{M} from s_0 is

$$s_0 \xrightarrow{(P, \text{Alice}, \text{Car})} s'_1 \xrightarrow{(S, \text{Alice}, \text{Bob})} s'_2 \xrightarrow{(S, \text{Alice}, \text{Don})} s'_3$$

with in particular

$$s'_2 \left\{ \begin{array}{ll} \text{Alice:} & \text{bonus} = 270 \\ \text{Bob :} & \text{bonus} = 130 \quad \text{sponsor} = \text{Alice} \\ \text{Car :} & \text{buyer} = \text{Alice} \quad \text{value} = 900 \\ \text{Don :} & \text{bonus} = 50 \quad \text{sponsor} = \text{Alice}. \end{array} \right.$$

In this execution, the rule instance $(S, \text{Alice}, \text{Don})$ is considered in s'_2 , where it is applicable. This contrasts with the previous execution, in which the rule instance is considered in state s_1 , where it is not applicable.

5.2 Refraction-Based Executions of the Rule Program

Let us consider the same rule program \mathcal{R} on a subset $\mathcal{M}' = \{\text{Alice}, \text{Bob}, \text{Car}\}$ of the working memory \mathcal{M} . Let us consider an execution of \mathcal{R} on \mathcal{M}' from the initial state σ_0 , equal to the restriction of s_0 to \mathcal{M}' , that is:

$$\sigma_0 \begin{cases} \text{Alice: } & \textit{bonus} = 230 \\ \text{Bob : } & \textit{bonus} = 100 \quad \textit{sponsor} = \text{Alice} \\ \text{Car : } & \textit{buyer} = \text{Alice} \quad \textit{value} = 900. \end{cases}$$

Let us note that Alice is a sponsor of Bob and the buyer of the car in state σ_0 , and that the actions of none of the rules in \mathcal{R} can change this. This implies that, per the definitions of Sect. 2.1, the only rule instances that can be applicable in any state of an execution of \mathcal{R} on \mathcal{M}' from σ_0 are $(P, \text{Alice}, \text{Car})$ and $(S, \text{Alice}, \text{Bob})$. In addition, the rule instance $(P, \text{Alice}, \text{Car})$ is applicable in σ_0 , and since the actions of none of the rules in \mathcal{R} can change this, it shall remain applicable in any state of an execution of \mathcal{R} on \mathcal{M}' from σ_0 . As a consequence, these executions would never terminate, unless at some point the control strategy did refrain from choosing $(P, \text{Alice}, \text{Car})$, even if it were the sole applicable rule instance. This is the role of the eligibility strategy, as illustrated below.

By definition, all rule instances are eligible in the initial configuration. The selection strategy will hence apply to the rule instances that are applicable in the initial state σ_0 , namely:

$$A_{\sigma_0} = \{(P, \text{Alice}, \text{Car}), (S, \text{Alice}, \text{Bob})\}.$$

Assume that the selection strategy chooses to apply $(S, \text{Alice}, \text{Bob})$ in σ_0 . This causes a transition by this rule instance from $\langle \mathcal{I}(\mathcal{R}, \mathcal{M}), \sigma_0 \rangle$ to $\langle E_1, \sigma_1 \rangle$, where the state σ_1 results from the application of $(S, \text{Alice}, \text{Bob})$ in σ_0 , that is:

$$\sigma_1 \begin{cases} \text{Alice: } & \textit{bonus} = \mathbf{180} \\ \text{Bob : } & \textit{bonus} = \mathbf{130} \quad \textit{sponsor} = \text{Alice} \\ \text{Car : } & \textit{buyer} = \text{Alice} \quad \textit{value} = 900 \end{cases}$$

and the set E_1 is computed by first removing $(S, \text{Alice}, \text{Bob})$ from $E_0 = \mathcal{I}(\mathcal{R}, \mathcal{M})$, and then adding all the rule instances that are not applicable in σ_1 . Since Alice has less than 200 bonus points in σ_1 , the rule instance $(S, \text{Alice}, \text{Bob})$ is not applicable in σ_1 , and $E_1 = \mathcal{I}(\mathcal{R}, \mathcal{M})$.

In configuration $\langle E_1, \sigma_1 \rangle$, we have $A_{\sigma_1} \cap E_1 = \{(P, \text{Alice}, \text{Car})\}$. Therefore, a transition to $\langle E_2, \sigma_2 \rangle$ occurs by the application of $(P, \text{Alice}, \text{Car})$, which results in

$$\sigma_2 \begin{cases} \text{Alice: } & \textit{bonus} = \mathbf{270} \\ \text{Bob : } & \textit{bonus} = 130 \quad \textit{sponsor} = \text{Alice} \\ \text{Car : } & \textit{buyer} = \text{Alice} \quad \textit{value} = 900. \end{cases}$$

The set E_2 is computed by removing $(P, Alice, Car)$ from E_1 , and then adding all the rule instances that are not applicable in σ_2 . Since $(P, Alice, Car)$ is applicable in σ_2 , it is not added back in E_2 , and we have

$$\begin{aligned} E_2 &= \mathcal{I}(\mathcal{R}, \mathcal{M}) \setminus \{(P, Alice, Car)\} \\ A_{\sigma_2} &= \{(P, Alice, Car), (S, Alice, Bob)\} \\ A_{\sigma_2} \cap E_2 &= \{(S, Alice, Bob)\}. \end{aligned}$$

Note that $(S, Alice, Bob)$ was not applicable in σ_1 but, in the transition to σ_2 , Alice's bonus has been increased to over 200 points, hence making $(S, Alice, Bob)$ applicable. As this rule instance has been inapplicable since its latest application, it must not be discarded by refraction: indeed, we have $(S, Alice, Bob) \in E_2$. The fact that $(S, Alice, Bob)$ is given the opportunity of being applied in σ_2 contrasts with the sequential execution, where this opportunity was denied in state s_2 .

The execution of \mathcal{R} continues with a second application of $(S, Alice, Bob)$, and results in $\langle E_3, \sigma_3 \rangle$ with

$$E_3 = E_2 \setminus \{(S, Alice, Bob)\} \cup \{R \in \mathcal{I}(\mathcal{R}, \mathcal{M}) \mid R \text{ is not applicable in } \sigma_3\}.$$

In this configuration, $(P, Alice, Car)$ is not included in E_3 since it is applicable in σ_3 . On the other hand, $(S, Alice, Bob)$ cannot be in $A_{\sigma_3} \cap E_3$ since either it is not applicable in σ_3 , or it is and is thus not eligible.

As there is no rule instance that is both applicable and eligible in $\langle E_3, \sigma_3 \rangle$, the transition rule \textcircled{II} cannot be applied and the execution of \mathcal{R} ends.

5.3 Discussion

As can be seen on the example discussed in this section, the set of eligible rule instances in each configuration is easier to compute in a sequential execution than in a refraction-based one. This explains why adopting a sequential execution semantics enables rule engines to execute rule programs faster.

On the other hand, this example also demonstrated that the refraction-based semantics gives more opportunity to rule instances to execute, whereas the sequential mode imposes a stricter control. This can be seen as an advantage of Rete-like execution, as this semantics seems more natural. However, some BRMS users regard this richer semantics as less predictable, and appreciate the greater control provided by the explicit ordering of rules of the sequential execution mode, especially with large rule programs as can for example result from the automatic translation of database tables into rules.

6 Conclusion

Business Rule Management Systems (BRMS) provide business applications with the ability to externalize part of their logic as rule programs. For a long time, these rule programs have been executed with the semantics linked to the Rete

algorithm. More recently however, an alternative to Rete has emerged, known as sequential, with a specific semantics.

In this paper, we give a formal description of the execution of rule programs by BRMS. In this description, we isolate the handling of rule eligibility in the control strategies of rule engines. We complete our formal description with the expression of selection and eligibility strategies for the Rete algorithm and for the sequential execution mode.

Finally, we illustrate our formalism with both a sequential and a refraction-based execution of an example rule-based application.

References

1. Baralis, E., Widom, J.: An algebraic approach to rule analysis in expert database systems. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB, pp. 475–486. Morgan Kaufmann (1994)
2. Berstel, B., Leconte, M.: Using constraints to verify properties of rule programs. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2010, pp. 349–354. IEEE Computer Society (2010)
3. Brownston, L., Farrell, R., Kant, E., Martin, N.: Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley, Boston (1985)
4. de Bruijn, J., Rezk, M.: A Logic Based Approach to the Static Analysis of Production Systems. In: Polleres, A., Swift, T. (eds.) RR 2009. LNCS, vol. 5837, pp. 254–268. Springer, Heidelberg (2009)
5. Cheng, A.M.K., Tsai, H.-Y.: A graph-based approach for timing analysis and refinement of OPS5 knowledge-based systems. *IEEE Transactions on Knowledge and Data Engineering* 16(2), 271–288 (2004)
6. Cirstea, H., Kirchner, C., Moossen, M., Moreau, P.É.: Production systems and Rete algorithm formalisation. Research report, LORIA, Nancy (September 2004), <http://hal.inria.fr/inria-00280938/PDF/rete.formalisation.pdf>
7. Corticon Business Rules Management System, <http://www.corticon.com/Products/Business-Rules-Management-System.php>
8. Damásio, C.V., Alferes, J.J., Leite, J.: Declarative Semantics for the Rule Interchange Format Production Rule Dialect. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 798–813. Springer, Heidelberg (2010)
9. Fages, F., Lissajoux, R.: Sémantique opérationnelle et compilation des systèmes de production. *Revue d’Intelligence Artificielle* 6(4), 431–456 (1992)
10. Fair, Isaac, and Company: High-volume batch processing with Blaze Advisor. Computer World U.K. (March 2007), <http://www.computerworlduk.com/white-paper/business-process/5092/high-volume-batch-processing-with-blaze-advisor/>
11. FICOTM Blaze Advisor ®, <http://www.fico.com/en/Products/DMTools/Pages/FICO-Blaze-Advisor-System.aspx>
12. Floyd, R.W.: A descriptive language for symbol manipulation. *Journal of the ACM* 8(4), 579–584 (1961)
13. Forgy, C.: OPS5 User’s manual. Tech. Rep. CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh (July 1981)

14. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence* 19(1), 17–37 (1982)
15. Hanson, E., Hasan, M.S.: Gator: An optimized discrimination network for active database rule condition testing. Tech. Rep. TR93-036, University of Florida (1993)
16. IBM Websphere Operational Decision Management, <http://www.ibm.com/software/websphere/products/business-rule-management>
17. IBM: IBM Websphere Operational Decision Management v7.5 User's Manual (2011), <http://publib.boulder.ibm.com/infocenter/dmanager/v7r5/>
18. McCoy, D.W., Sinur, J.: Achieving Agility: The Agile Power of Business Rules. Gartner (2006), http://www.gartner.com/DisplayDocument?doc_cd=138218
19. Mettrey, W.: A comparative evaluation of expert system tools. *Computer* 24, 19–31 (1991)
20. Miranker, D.P.: TREAT: A better match algorithm for AI production systems. In: *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI 1987*, vol. 1, pp. 42–47. AAAI Press (1987)
21. Oracle Business Rules, <http://www.oracle.com/technetwork/middleware/business-rules/overview>
22. Red Hat: JBoss Enterprise BRMS, <http://www.jboss.com/products/platforms/brms>
23. Red Hat: Drools Expert User Guide (2010), http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-expert/html_single/index.html#d0e2086
24. Rosenthal, D.: Adding meta rules to OPS5: A proposed extension. *ACM SIGPLAN Notices* 20(10), 79–86 (1985)
25. de Sainte Marie, C., Hallmark, G., Paschke, A.: Rule Interchange Format, Production Rule Dialect. Recommendation, W3C (2010), <http://www.w3.org/TR/rif-prd/>
26. SAP NetWeaver Business Rules Management, <http://www.sap.com/platform/netweaver/components/brm/index.epx>
27. Schmolze, J.G., Snyder, W.: Detecting redundancy among production rules using term rewrite semantics. *Knowledge-Based Systems* 12(1–2), 3–11 (1999)
28. Soloway, E., Bachant, J., Jensen, K.: Assessing the maintainability of XCON-in-RIME: Coping with the problems of a VERY large rule-base. In: *AAAI*, pp. 824–829 (1987)
29. Taylor, J., Raden, N.: *Smart (enough) systems: How to deliver competitive advantage by automating the decisions hidden in your business*. Prentice Hall, Upper Saddle River (2007)

Bringing OWL Ontologies to the Business Rules Users

Adil El Ghali^{1,2}, Amina Chniti^{2,3}, and Hugues Citeau²

¹ LUTIN UserLab, 30, avenue Corentin Cariou, 75019 Paris, France
adil@elghali.name

² IBM CAS France, 9 rue de Verdun, 94253 Gentilly, France
{amina.chniti,hugues.citeau}@fr.ibm.com

³ INSERM UMRS 872, Eq 20, 15, Rue de l'école de médecine, 75006, Paris, France

Abstract. Ontologies are known to be suitable to represent business knowledge. However, in the Business Rules community the business models are usually represented using object models (OM). Many of the existing Business Rules Management Systems (BRMS) allow the Business Users to represent Business Object Models in their own proprietary languages. Some work has been done in the last years to bridge the gap between the ontologies and the Business Rules. A pragmatic approach consist in projecting ontologies into the Object Models used by the BRMS, to ease the use of ontologies by the Business Users. The main issue with this approach is that the expressive power of the targeted Object Model is not enough to cope with the content of the ontology. Hence, the translation loses some of the information contained in the ontology, such as axioms. The aim of this paper, is to go a step further using this approach by translating some of the axioms defined in an OWL ontology into Business Rules. This translation brings at least two benefits: (i) it allow the Business Users to understand better the content of the Ontology by having some of its axioms in the rule language they are used to. (ii) at the run-time level, the translated axioms will be handled by the rule engine. We explain the basic mechanism of this translation and detail its implementation in the JRules BRMS system.

Keywords: Ontologies, Business Rules, OWL, RIF-PRD.

1 Introduction

Ontologies are more and more used to represent domain knowledge in business applications. The flexibility of ontologies make it possible to define the domain knowledge across different applications. On the other hand, business rules are an established framework for defining a large class of business applications. Many software vendors develop business rules management systems (BRMS) and a lot of complex decision centered business applications are built using them. Those BRMS use in the majority of cases object oriented models to formalize the domain knowledge.

These so called Business Object Models, are in general easy to use thanks to the existing tools to manipulate them, and they are well understood by the business users to represent the domain knowledge for the Business Rules application, however their expressive power is not as good as the OWL ontologies [8], this weakness requires the users to hand-write parts of their business model using operative rules. The use of OWL ontologies to represent business models will reduce the need of having rules that encode parts of the models, these rules are then replaced by OWL axioms that constraint the model. Meanwhile, the business users writing rules are not always familiar with OWL and do not have a good understanding of the impact of axioms on the execution of their rules.

The aim of this work is to provide a simple mechanism to better exploit the power of OWL in business rules applications, while preserving the rules developers understanding of the implications of OWL axioms on their application. The method we present in this paper is based on a loosely-coupled approach between Rules and Ontologies, such as in [6]. In our approach the execution of Rules that use OWL Ontologies to represent domain knowledge is carried by two engines a Rules Engine and an OWL reasoner. The basic idea is to map the OWL ontology into a Business Object Model that can be used by the Rule Engine, and to allow business users to author and execute their rules on this model [4], an OWL reasoner is called by the Rule Engine whenever necessary to carry out reasoning on the model. However, since the representation of the world used by a Rule Engine is Object-Oriented while the OWL ontologies model the world is Fact-based, the impedance mismatch between the two make it difficult to achieve a full mapping between the two at the model level [3], and may confuse the business users writing the rules since they are used to Object-Oriented models and not familiar with Ontologies.

We propose to tackle these issues with a method based on the idea of OWL2RL in RIF [7] that moves some of the OWL reasoning capabilities to the Rule Engine by encoding OWL axioms using rules, and by integrating these reasoning rules into the BRMS used by business users to author, manage and execute their rules. The intuition behind this method is that the OWL ontology is translated, in addition of a Business Object Model, into a set of reasoning rules that encode some of the OWL reasoning capabilities using the Rule Engine. These reasoning rules, presented in a business language, are also provided to help the business user that develop the rule-based application to understand the OWL axioms. The translation of the ontology into a Business Object Model and a set of reasoning rules can be either complete or partial, in the first case the OWL reasoner is not needed anymore and the Rule Engine carry out all the execution tasks, in the second case, the OWL reasoner will still be necessary to carry out the parts of the reasoning that are not covered by the reasoning rules.

We implemented this method on the top of the IBM WebSphere ILOG JRules a widely used BRMS.

The paper is organized as follow, we start by introducing, in section [2], how we add OWL support to the JRules BRMS, we then present the mechanic behind

the translation of OWL axioms into rules, in section 3, and illustrates it with a small example in section 4. We discuss some of the implications of the method on the execution of the rules in section 5, before concluding and sketching the possible evolutions and perspectives.

2 Using OWL in JRules BRMS

JRules offers an infrastructure that enables to author - in a controlled natural language -, to manage and to execute business rules. The JRules OWL plugin exploit this infrastructure to import OWL ontologies within JRules and to perform a mapping of OWL concepts (TBox) into the JRules formalism, the Business Object Model (BOM) (see Section 2.1), which is used to represent the concepts of the domain. Thus, when we import an OWL ontology within JRules, the BOM is automatically generated and the functionalities offered by the BRMS (i.e rule authoring, rule execution and rule maintenance) can be used over the content of the OWL ontology (see Figure 1).

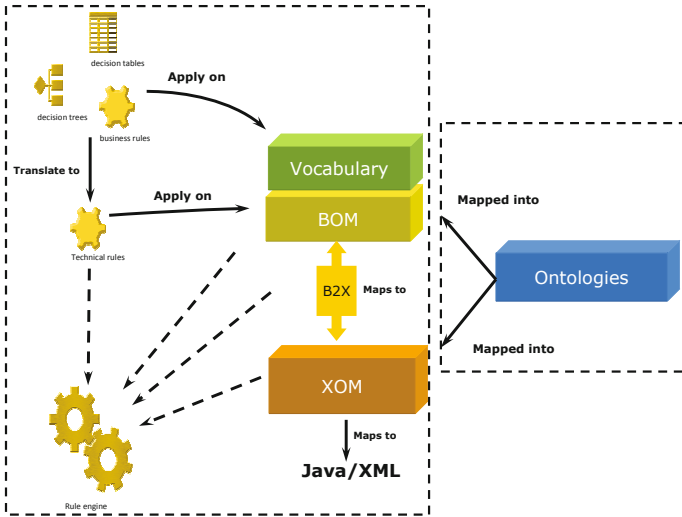


Fig. 1. Adding basic OWL capabilities to JRules

2.1 IBM WebSphere ILOG JRules BRMS

The management of business policies becomes a complex task due to their implementation into a hard-coded application and their continuous evolution. Business Rule Management System (BRMS), provides solutions to make this management

more efficient by externalizing the business logic from the hard-coded application using Business Rules, which enables business users to manage business policies with limited dependence on the IT department.

JRules is a BRMS, as such, it provides the means to author, manage, and execute business rules. The authoring of business rules into JRules is performed over the BOM which can be generated from an eXecutable Object Model (XOM) that enables the rule execution. In the following we will introduce the BOM and the XOM which are the main components for authoring and executing rules within JRules.

Business Object Model (BOM). The BOM is an object model that represents the concepts of a given business. It defines the entities, actions and the vocabulary used in business rules. A BOM contains a set of classes grouped into packages and each class has a set of attributes and methods, which the rules act on. It is generated from the XOM and is then verbalized. The verbalization consists of generating a controlled natural language vocabulary (VOC) which enables to edit the business rules.

eXecutable Object Model (XOM). The XOM is the model enabling the execution of rules. It references the application objects and data, and is the base implementation of the BOM. The XOM can be built from compiled Java classes (Java execution object model) or XML Schema (dynamic execution object model). Through the XOM, the rule engine can access application objects and methods, which can be Java objects or XML data. At runtime, rules that were written via the BOM are executed over the XOM.

Business Rules. From a business perspective, a business rule is a precise statement that describes, constrains, or controls some aspect of your business. From the IT perspective, business rules are a package of executable business policy statements that can be called from an application.

In Jrules, business rule is an expression of a business policy in a form of “If-Then” statements that are understandable by a business user and executed by a rule engine. For instance:

*IF the age of the client is between 18 and 25
THEN set the insurance ratio to 125*

This controlled language, called Business Action Language (BAL), is compiled into a lower-level technical language. The “business layer” is composed of two models supporting the definition of the rules. The business objects of the domain (*client*, *age*), which are represented in the BOM and the vocabulary model (VOC), which add a layer of terminology on top of the BOM (“*the client*”, “*the age of the client*”). This vocabulary, introduced with the VOC is in turn used to compose the text of the rules [5].

2.2 Authoring Business Rules over Ontologies : OWL to BOM Mappings

As described in Section 2.1 the BOM, is the main component for authoring rules in JRules. To enable authoring business rules over OWL ontologies, we performed a mapping of OWL construct into the BOM. This mapping enables the automatic generation of the BOM when importing an OWL ontology within JRules.

Due to the differences in knowledge representation conventions between the BOM and OWL, there are some OWL constructs that cannot be mapped into the BOM such as `owl:disjointWith`, `owl:complementOf`, `owl:someValuesFrom...`. The mapping from OWL to BOM is achieved as follow :

Classes: An OWL class is mapped into a BOM Class. The hierarchical relations are mapped into the BOM using the subclass relation and, as the BOM supports multiple inheritance, this information is also preserved;

Properties: An OWL property¹ is mapped into an attribute of a BOM class. Functional properties are mapped to single attributes and multi-valued properties are mapped to multiple cardinality attributes. The class of an attribute corresponds to the domain of the corresponding property, and its type corresponds to the range of the property.

Nevertheless, a property may have a null or multiple domain (range, respectively). These cases are mapped as follows:

- *Null domain* : the attribute is added to all the root classes, (i.e which inherit directly from `owl:Thing`);
- *Multiple domain* : the attribute is added to all the classes corresponding to the set of the domains;
- *Null range* : the type of the attribute is inferred as follow :
 - if the property has an equivalent property then the type of the attribute will be the range of the equivalent property;
 - if the property has an inverse property then the type of the attribute will be the domain of the inverse property;
 - otherwise the type of the attribute will Object.

Restrictions:

- `owl:cardinality` and `owl:maxCardinality` restrictions: when an attribute's such restrictions are equal to 1, it is mapped into a single-valued attribute; otherwise, it is mapped into a multiple-valued attribute;
- `owl:allValuesFrom` restriction: the type of the attribute is the class defined on the restriction;
- `owl:oneOf` restriction: Static values corresponding to the values defined on the collection are attached to the class.

¹ ObjectProperties and DataTypeProperties are handled the same way, since in the BOM there is no distinction between the two.

The table [1](#) summarize the mapping.

Table 1. OWL to BOM Mapping

OWL	BOM
owl:Class ?A	Class A
?B rdfs:subClassOf ?A	Class B extends A
?C owl:intersectionOf(?A,?B)	Class C extends A,B
?C owl:unionOf(?A,?B)	Class A extends C and Class B extends C
?A owl:oneOf {x, y, z}	Class A {domain {'x', 'y', 'z'};}
?A owl:equivalentClass ?B	Keep only A, and references to B are reported to A
rdf:Property ?P(?A,?B)	Class A {B[] P};
P rdfs:subPropertyOf P'(?A,?C)	Class A {B[] P; C[] P';}
P' owl:equivalentProperty P	Class A {B[] P; B[] p';}
P' owl:inverseOf P	Class B {A[] P};
P owl:functionalProperty	Class A {B P};
P.cardinality = 1	Class A {B P};
P.maxCardinality = 1 on P	Class A {B P};
P owl:allValuesFrom C	Class A {C[] P};

2.3 Executing Business Rules over OWL Ontologies

The process of executing business rules in JRules consists of several steps. Business rules, authored in a controlled natural language (BAL) are translated into executable rules, which are written in a formal technical rule language ILOG Rule Language (IRL). During this translation, the references to the BOM's classes and properties are translated to references into the XOM. When the input provided to JRules is a Java object model, the XOM is built from this model. But in our case, the input provided to JRules is an OWL model.

Before introducing OWL axioms, to execute business rules authored from ontologies, we perform a second mapping of OWL-to-BOM entities to a XOM using Jena.² Jena is a Java framework, including an API which allows to generate Java objects from the entities of the ontology. These Java objects then constitute the XOM.

The use of Jena provides an execution layer for the OWL ontologies. This execution layer provides inference mechanisms on this model and the mapping of OWL concepts, properties, and individuals to a Java object model.

2.4 Business Users Interactions with the System

A business scenario is described in the following based on personas that define different kinds of business users involved in building and using a rule-based application as defined in [\[1\]](#).

² <http://jena.sourceforge.net/>

The scenario stages three personas, Marc, Alice and Joana. Marc is the business analyst. He acts as a bridge between business and IT departments in order to design the Business Rules Tools. His mission is to formalize the business knowledge, so that it can be transformed into IT requirements. Alice is the domain expert. She authors the rule of her domain, understands their formalization and knows how to use BRMS. Joana is the operational user. She is the user of the rule application. The rules allow her to realize business operational tasks, she sees their execution but can't their structure.

As shown in figure 2, the first step is achieved by Marc who imports the OWL ontology in JRules and a BOM is automatically generated. Then Marc checks the verbalization for this BOM proposed in the application. He can modify some terminology proposed if needed. The Business vocabulary is then used by Alice to author business rules. Then, Alice edits the rules with the business vocabulary used in the company. She knows well the business policies of her domain and has been trained for editing rules in the business rule application. Finally, Joana uses the authored rules to realize business operational tasks she is in charge of. If she sees the rules executed she can recognize business vocabulary used in the company.

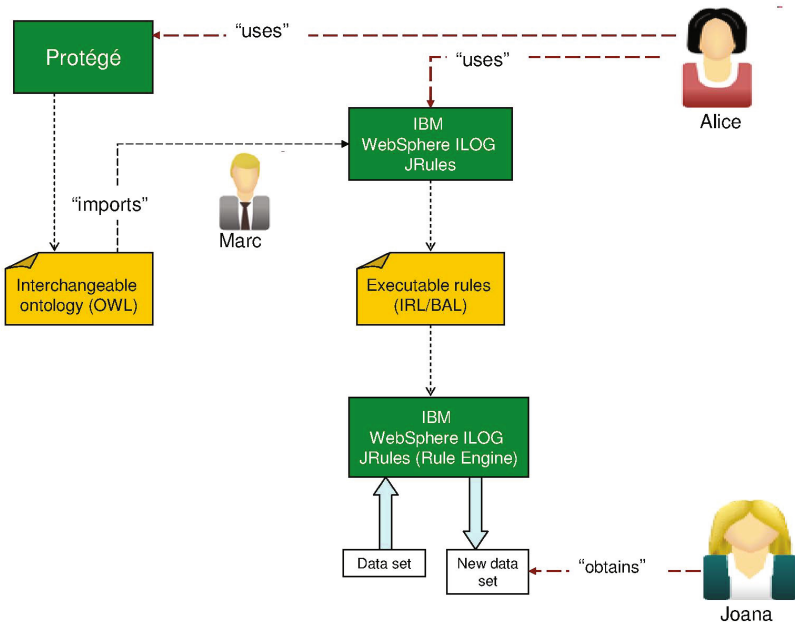


Fig. 2. Business Scenario

3 OWL Axioms as Rules

In our loose coupling approach, the Rule Engine is expected to delegate the reasoning tasks related to classification and navigation to the OWL reasoner. Our goal is to move some of the reasoning capabilities to the Rule Engine side. The production rules for advanced reasoning are added to the applicative rules as higher priority rules. Those rules are adding reasoning related assertions such as typing assertions to the OWL ontology. The OWL reasoner will take into account those new reasoning assertions the next time it is asked to answer a classification or a navigation query.

The first issue to solve here is to know how to express the reasoning production rules. The OWL2RL ruleset [7] is providing us with these reasoning rules. Those rules may be derivation rules or production rules. Forward chaining is sufficient to process those rules. In the case of derivation rules, the rule language cannot be as basic as it seems. Conjunction in the head must be supported because some OWL2RL rules are providing more than one derivations at once. Sometimes, those derivations are even not of a fixed size. Moreover, the matching of lists of arbitrary length must also be supported in the body. Some rewriting of the OWL2RL may take place to normalize it to simpler derivation rules. But from a practical perspective, this rewriting will give simpler rules only if the ruleset is instantiated on a fixed user vocabulary. This instantiation will produce more reasoning rules than the original OWL2RL rules. The OWL2RL rules are only manipulating RDF triples. The reason is that on many places, variables are introduced as placeholders that denote classes or properties. OWL2RL rules can be expressed differently. But either the rules language must support variables for classes and properties, or the ruleset must be rewritten in the context of a fixed user vocabulary for classes and properties [2].

Let us illustrate this with an example of an OWL2RL production rule that express the `equivalentClass` axiom in OWL, in [7] this rule is using RDF-based semantic of OWL :

```
production rule EQAxiomRule {
  when {
    Triple(?c, rdf:type, ?cc);
    Triple(?cc, owl:equivalentClass, ?cd);
  }
  then {
    insert Triple(?c, rdf:type, ?cd);
  }
}
```

When translated into a rule language that accepts variables for classes and properties, it will be rewritten as follow :

```
production rule EQAxiomRule_higher {
  when {
    ?cc(?c);
```

```

    owl:equivalentClass(?cc,?cd);
  }
  then {
    insert ?cd(?c);
  }
}

```

But in order to be presented to a business user, or in a rule language that do not accept variables for classes and properties, the rule should be instantiated into a set of rules for all the classes that are concerned by the `owl:equivalentClass` axiom in the ontology, so for each classes `CC` and `CD` where `owl:equivalentClass(CC,CD)` we obtain a reasoning rule `EQAxiomRule_CD`:

```

production rule EQAxiomRule_CD {
  when {
    CC(?c);
    owl:equivalentClass(CC,CD);
  }
  then {
    insert CD(?c);
  }
}

```

The reasoning ruleset is added to the applicative ruleset, these rules must have a higher priority for all the reasoning steps to be properly taken into account in the applicative rules. At this point, we still want the OWL reasoner to perform the subset of the reasoning that is not covered by the produced ruleset. But this introduce a coupling issue that we need to solve. The state of the OWL ontology will need to be shared by both the OWL reasoner and the Rule Engine. The OWL reasoner will expect this state to be available as facts or triples, But an object-oriented Rule Engine will not be able to deal with facts so easily. The cycle that is expected to take place is that the Rule Engine is matching the content of the OWL ontology, since it also takes care of advanced reasoning, the content is not only the A-Box but also the T-Box. The advanced reasoning production rules add derived assertions to the OWL ontology. The OWL reasoner must be aware of those new assertions. The Rule Engine itself must also properly update its matching state to take into account the new OWL assertions.

The state synchronization can be added to the reasoning rules directly using `rdf:type` assertions, when it is using RDF-based semantic. On each firing of these rules the action part will achieve two actions, one for the working memory of the Rule Engine, and the other for the OWL ontology:

```

production rule Sync_EQAxiomRule {
  when {
    Triple(?c,rdf:type,?cc);
    Triple(?cc,owl:equivalentClass,?cd);
  }
}

```



```

then {
  // 1- Add: Triple(?c,rdf:type,?cd) to the OWL ontology
  //     for the OWL reasoner
  // 2- Add: Triple(?c,rdf:type,?cd) to the working memory
  //     for the PR engine
}
}

```

On a Rule Engine that does not support variables for classes and properties, the actions are also duplicated to deal with the synchronization :

```

production rule Sync_EQAxiomRule_CD {
  when {
    CC(?c);
    owl:equivalentClass(CC,CD);
  }
  then {
    // 1- Add: CD(?c) to the OWL ontology for the OWL reasoner
    // 2- Add: CD(?c) to the working memory for the PR engine
  }
}

```

Now that we know how to obtain the reasoning ruleset for an ontology, we need to produce these rules in a user-friendly format. Our translation produce the reasoning rules in the BAL language taking advantage of the verbalizations contained in the ontology for classes and properties. The rules presented to the business user will have the following form (for the EQAxiomRule_CD) :

IF $_CC$ *is equivalent to* $_CD$ *AND the type of* c *is* $_CC$
THEN set the type of c *to* $_CD$

We create for each BOM entity translated from the OWL ontology an automatic variable that is used to designate it in the reasoning rules.

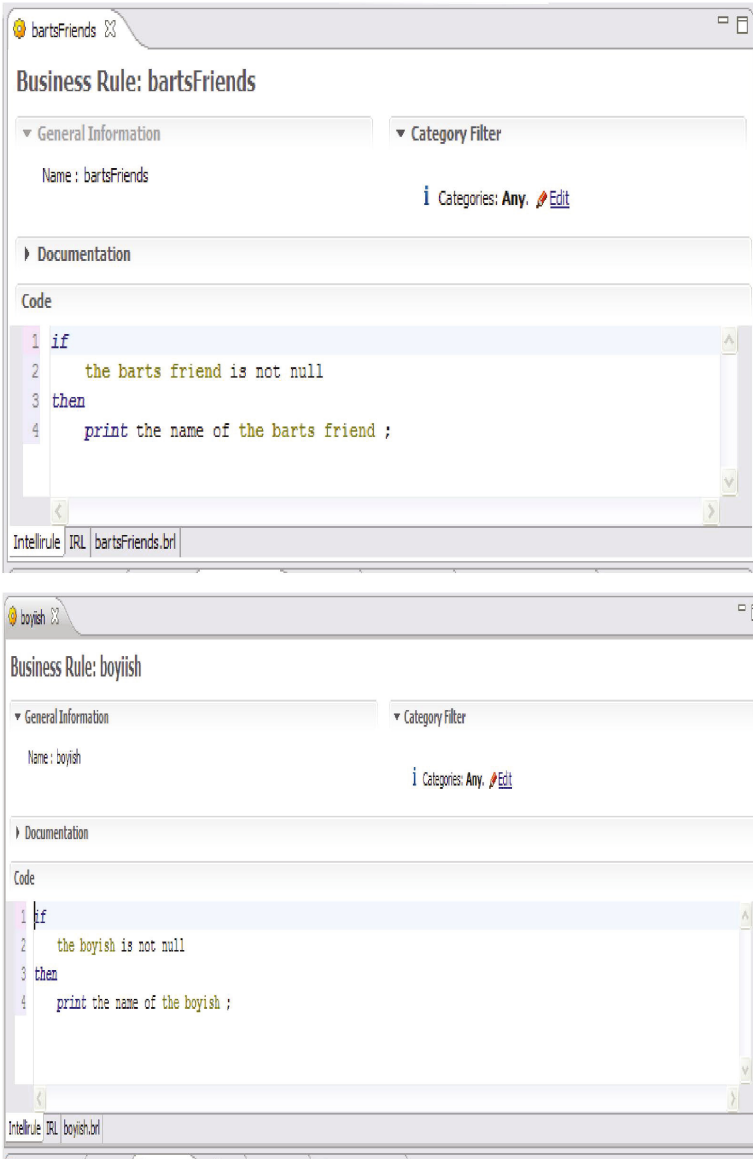
4 The Simpsons Example

To illustrate the work we will use the Simpson ontology. In this ontology :

1. the concept *Boyish* is defined using the restriction *owl:allValuesFrom* such as *hasFriend* *only* *Boy* where *Boy* is a concept;
2. the concept *BartsFriend* is defined using the restriction *hasValue* such as *hasFriend* *value* *Bart* where *Bart* is an individual of the concept *Boy*.

The rule set of our rule project contain :

1. *bartsFriends* rule that lists the name of person that have *Bart* as a friend;
2. *boyish* rule that lists the name of all persons that have only boys as friends. (see Figure 3)

**Fig. 3.** Simpson Rules

These rules cannot be directly executed as the rule engine cannot reason on the `owl:hasValue` and the `owl:allValuesRestriction` restrictions. To resolve this problem we implement the so called *reasoningRules* that enables to set the type of a *BusinessThing* depending its restriction. *BusinessThing* is a super concept of all the business concept of the ontology and is a sub concept of `owl:Thing`. It define a property *type* (*BusinessThing*, *BusinessThing*) that represents the type of a business thing. To resolve the problems discussed above the following, *reasoningRules*, will be executed before the *Simpson rules* (see Figure 4).

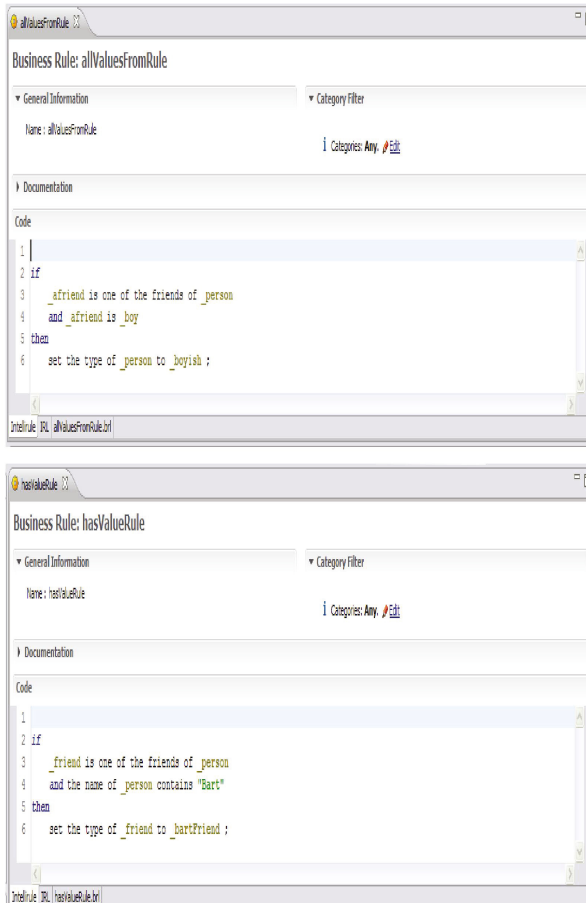


Fig. 4. Reasoning Rules

The rule *allValuesFromRule* tests if the friends list of a person contains only boys then the type of the person is set to *Boyish*. The rule *hasValueRule* tests if the list of friends of a person contains *Bart* then the type of the person is set to *BartsFriend*. The execution process of the rules is orchestrated by a rule flow.

5 Discussion

In the previous sections, we focused in the practical issues of translating the axioms of an OWL ontology into business rules. But a fundamental issue has not been discussed, namely the case when the production rule engine is object-oriented and not fact based. The OWL reasoner is definitely fact based. But a state of practice production rule engine is usually object-based. It means that it is dedicated to objects, the nodes of the runtime graph. But it is not dedicated to relations, the edges of the runtime graph. Objects are related through functions. They are not related through relations. Practically, it means that only one navigation path is supported from the subject (aka this) to another object. The engine does not know how to follow the inverse navigation path. The inverse navigation path has to be provided explicitly in the object model. It also has to be maintained consistent with its dual counterpart manually. Moreover, the engine is not able to deal with multiplicity. A subject is pointing to a single other object. Collections have to be added manually to deal with multiplicity. On one hand, the object-oriented approach is very efficient at runtime. But on the other hand, its level of expressiveness is weaker when compared to the fact-based approach. In a fact-based approach, many subjects may be in relation with many subjects by default. Hence, an OWL reasoner or a fact based production rule engine have all the facilities to handle inverse navigation paths and multiplicity as built-in constructs. What is looking like a detail has in fact a huge impact on the design of the knowledge model and of the production rules. What is automated in a fact-based approach needs to be manually added to an object-oriented approach. And in the context of a coupling with an OWL ontology, there need to be two different representations of the same knowledge, one fact-based for the OWL reasoner and one object-based for the production rule engine. Some additional synchronization needs to take place.

The object-oriented version of a reasoning production rule using RDF triples:

```
production rule OO_Sync_EQAxiomRule {
  when {
    Triple(subject == ?c; predicate == rdf_type; object == ?cc);
    Triple(subject == ?cc; predicate == owl_equivalentClass;
      object == ?cd);
  }
  then {
    Triple triple = new Triple(?c,rdf:type,?cd);

    ontology.addTriple(triple); // for the OWL reasoner
    insert triple; // for the PR engine
  }
}
```

The object-oriented version of a reasoning production rule accepting variables for classes and properties:

```
production rule OO_Sync_EQAxiomRule_higher {
  when {
    ?c:?cc();
    owl:equivalentClass(first == ?cc; second == ?cd);
  }
  then {
    ontology.addClassAssertion(?c,?cd); // for the OWL reasoner
    insert (?cd)?c; // for the PR engine
  }
}
```

The object-oriented version of a reasoning production rule instantiated on a fixed user vocabulary:

```
production rule OO_Sync_EQAxiomRule_CD {
  when {
    ?c:CC();
    owl:equivalentClass(first == CC; second == CD);
  }
  then {
    ontology.addClassAssertion(?c,?cd); // for the OWL reasoner
    insert (CD)?c; // for the PR engine
  }
}
```

The benefit of using object-oriented reasoning rules is that it becomes possible to abstract them to business rules using the object-oriented legacy tooling so that business user can watch and understand what is going on during reasoning. The business object-oriented reasoning production rules are also perfectly aligned with the business applicative production rules.

6 Conclusions

The method presented in this paper allows the users of the Business Rules community to use OWL ontologies to represent the domain knowledge in their applications. The translation of an OWL ontology into a Business Object Model, and a set of Reasoning Rules - that represent the axioms of the ontology -, offers to the Business Users a convenient way to easily exploit the power of the OWL language even without a prior knowledge of OWL. This method will, in our point of view, ease the adoption of OWL as formalism to represent the domain knowledge in Business Rules applications. But it will also, help the Business Users to make better applications since they will have a clearer idea on the separation between the what can be represented in the model and what can be represented using the rules.

The actual implementation of the method is still in an early stage of development, and we are working on improving many aspects of it. The first one is on the presentation of the Reasoning Rules to the Business Users, on this aspect, we are investigating how we could cluster these rules and present them efficiently to the users. The second aspect is on the execution side, where we still need to compare the respective performances of the OWL reasoner and the Rule Engine when dealing with the reasoning rules.

Acknowledgments. The work described in this paper has been partially supported by the European Commission under ONTORULE Project (FP7-ICT-2008-3, project reference 231875, <http://ontorule-project.eu>).

References

1. de Bonis, S., Bellino, C., El Ghali, A.: Final usability report: evaluation and conclusions. Tech. Rep. D2.5, ONTORULE project (2011)
2. de Bruijn, J.: RIF, RDF and OWL compatibility. Proposed recommendation. Tech. Rep., W3C (October 2009), <http://www.w3.org/TR/2010/PR-rif-rdf-owl-20100511/>
3. de Bruijn, J.: State-of-the-art survey of issues. Tech. Rep. D3.1, ONTORULE project (2009)
4. Chniti, A., Dehors, S., Albert, P., Charlet, J.: Authoring Business Rules Grounded in OWL Ontologies. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 297–304. Springer, Heidelberg (2010)
5. Del Fabro, M.D., Albert, P., Bézivin, J., Jouault, F.: Achieving Rule Interoperability Using Chains of Model Transformations. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 249–259. Springer, Heidelberg (2009)
6. Meditskos, G., Bassiliades, N.: Hoopo: A hybrid object-oriented integration of production rules owl ontologies. In: ECAI, pp. 729–730 (2008)
7. Reynolds, D.: OWL 2 RL in RIF. Tech. Rep., W3C (June 2010), <http://www.w3.org/TR/rif-owl-rl/>
8. Tomaiuolo, M., Turci, P., Bergenti, F., Poggi, A.: An Ontology Support for Semantic Aware Agents. In: Kolp, M., Bresciani, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2005. LNCS (LNAI), vol. 3529, pp. 140–153. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11916291_10

From Regulatory Texts to BRMS: How to Guide the Acquisition of Business Rules?*

Abdooulaye Guissé, François Lévy, and Adeline Nazarenko

Université Paris 13, Sorbonne Paris Cité
Laboratoire d'Informatique de Paris-Nord (LIPN), CNRS (UMR 7030)
F-93430, Villetaneuse, France

Abstract. This paper tackles the problem of rule acquisition, which is critical for the development of BRMS. The proposed approach assumes that regulations written in natural language (NL) are an important source of knowledge but that turning them into formal statements is a complex task that cannot be fully automated. The present paper focuses on the first phase of this acquisition process, the normalization phase that aims at transforming NL statements into controlled language (CL), rather than on their formalization into an operational rule base. We show that turning a NL text into a set of self-sufficient and independent CL rules is itself a complex task that involves some lexical and syntactic normalizations but also the restoration of contextual information and of implicit semantic entities to get a set of self-sufficient and unambiguous rule statements. We also present the SemEx tool that supports the proposed acquisition methodology based on the selection of the relevant text fragments and their progressive and interactive transformation into CL rule statements.

1 Introduction

Checking the conformance of a process with respect to regulations is a growing domain of application for business rule management systems (BRMS). For instance, in order to export cars in various countries, car manufacturers have to satisfy safety and quality tests described in UNO regulations (*e.g.* 50 pages without annexes for the sole safety belts) and others constraints from the European and national authorities. Moreover, these regulations evolve over time (the UNO text has been modified 10 times between 2005 and 2009). Even if efficient rule systems are now able to exploit and maintain large rule bases, rule acquisition remains a bottleneck, and text-based rule acquisition is an important challenge.

We propose a method for the acquisition of rules and a tool, SemEx, which supports that method and guides the acquisition process. This approach relies

* This work was realized as part of the FP7 231875 ONTORULE project (<http://ontorule-project.eu>). We thank our partners for the fruitful discussions, especially to John Hall (Model Systems) for introducing us to the SBVR world and to Audi for the collaboration on their use case. We are also grateful to American Airline who is the owner of one of our working corpora.

on two strong assumptions. First of all, we believe that acquiring conformance rules from regulations cannot be fully automated. This is due to the complexity of human natural language (NL). The acquisition strategy that we propose relies on the cooperation of a domain expert and local automated processes. The expert controls the transformation of the regulation but automatic processes ease up the expert work. Second, we consider that it is difficult if not impossible to translate directly NL regulations into an operational rule bases expressed in formal language. We rather propose to decompose the formalization work into two main phases and to use controlled languages (CL) such as SBVR structured English (SBVR-SE¹) as an intermediate language. A domain expert designs a set of CL rules from the source NL regulations and this set of candidate rules is then passed on to a specialist of information technologies (IT) that formalizes the candidate rule base taking the characteristics of the final application and the constraints of the rule engine into account. We focus here on the first phase, the normalization of the source regulation into a set of candidate rules written in CL, rather than on the second phase, the transformation of the CL statements into rules, which has been more studied. We show how this normalization process can be divided into intermediary steps, which allows to decompose the expert work and to guide each step with specific helping tools.

This rule acquisition method and the associated SemEx tool have been developed as part of the ONTORULE project, which aim was to define an integrated platform for acquisition, maintenance and execution of business-oriented knowledge bases combining ontologies and rules. The work has been tested on two industrial use cases. The *AAAdvantage use case* aimed at developing a classification application to determine the benefits that an airline customer retention program member has earned over a given period. The business rule model had been designed from the documentation downloaded from the American Airlines (AA) web site, in particular the Terms and Conditions (5,744 words), which describes the membership statuses and the associated benefits. In the *Audi use case*, a rule application has been being defined to certify the conformance of Audi procedures with vehicle safety international regulation. The present experiment is based on the aforementioned UNO regulation. In each use case, the normalization process has been guided by a domain ontology that had been built beforehand.

Section 2 presents the state of the art. The normalization methodology and the architecture of the SemEx tool are described in Section 3. Section 4 details the framework that we propose for the progressive translation of texts into CL, showing the complexity of the involved linguistic and semantic transformations. Section 5 presents the results of that normalization processes in our use cases.

2 Related Works

BRMS are useful for propagating automatically the changes in the business of organizations into their information systems [1]. According to [11], the different forms of business rules can be seen as a continuous flow of models: the rules

¹ Semantic Business Vocabulary and Rules <http://www.omg.org/spec/SBVR/1.0/>

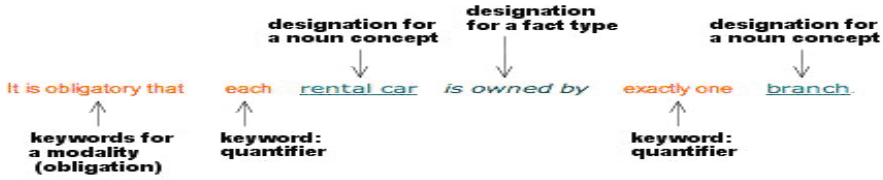


Fig. 1. SVBR SE business rule example (www.brcommunity.com)

evolve from an initial state (the rules are included in documents specifying the system), to a final state (they are formalized and integrated in information systems). However, some problems remain unsolved regarding 1) the acquisition of rules from specification documents, 2) their modeling in a formal language that enable their automation, 3) their integration in BRMS (storage, exploitation and maintenance).

The present work focuses on the first two points. This question, already raised by [3,11,9], concerns the transformation from informal to formal knowledge and the translation of text fragments written in NL into formal rules. This translation is difficult to automate, due to the complexity of NL and reduced expressivity of formal languages. Even the translation into SPARQL of LN queries, which are much simpler than texts, is acknowledged as a complex problem. To the best of our knowledge, only [8] considers a direct translation of legal texts but, after a parsing step, the abstract syntax trees are translated by hand into CTL².

Controlled languages have been proposed as intermediate languages in this translation process [18]. They allow to reformulate rules in a way that is still readable for the user and is easier to formalize than NL. In the Business Rules domain, CLs are used in Oracle Policy Modeling Suite³, in IBM SPARCLE policy workbench [4]. Other in use controlled languages have been described by RuleSpeak [16], by Attempto Controlled English (<http://attempto.ifi.uzh.ch/site/docs/>).

SBVR (Semantics of Business Vocabulary and Rules) can be seen as a synthesis of several efforts and a standard independent of English or any natural language. It has been accepted by the OMG (Object Management Group)⁴. We refer to the English version of SBVR CL, namely SBVR Structured English (SBVR-SE). SBVR relies on formulas (Figure 1) combining linguistic basic templates with logical, modal or quantification operators.

The NL to CL translation is a complex task, the automation of which has been rarely considered. Recently, [2] has proposed NL2SBVR⁵, a tool to automatically translate NL into SBVR-SE. According to the reported experiments, the complexity of the translation depends on the number of clauses that compose a sentence. Only so-called simple rules, composed of at most two clauses, are

² CTL is a modal temporal logic.

³ www.oracle.com/technology/products/applications/policy-automation

⁴ <http://www.omg.org>

⁵ <http://www.cs.bham.ac.uk/~isb855/nl2ocl/projects.html>

translated with a 80% success rate. This is the reason why, in SemEx, translation into CL and simplification go along for complex and long NL rules.

NL simplification has been studied to ease translation [17], human understanding (esp. in case of understanding disorders [13]), text summarization [10,7], foreign language learning (see [6] for a general presentation). Methods give a significant role to the lexical part, trying to stick to a privileged vocabulary, and to the brevity of sentences. Our transformation process (see Section 4) relies on the same methods but differs in its goal. The above works aims at preserving the discursive structure of texts while sometimes simplifying the information content, whereas, in business rule acquisition, the simplification of text should give a set of independent rule statements but preserve their meaning.

3 Normalization Process

3.1 Overall Approach

The normalization process takes a regulatory text as input and outputs a new "text" composed of a list of independent, self-sufficient rules. The rules are written in a language that is as controlled as possible and the set of rule forms a draft of a business rule model (the basis of a formal rule model). The proposed approach relies on the *selection* of the relevant text fragments (sentences or sequences of sentences) that convey rule information, and on their *normalization*, *i.e.* on their translation into CL. Both steps are difficult to handle automatically. The selection step calls for browsing facilities, since missing some important passages leads to a partial BR model. Only simple natural language statements can be automatically and reliability translated in CL. An interactive approach is therefore adopted, which consists in transforming step-by-step a regulatory text fragment into one or several independent rules written in controlled language.

This approach is illustrated on Figure 2. The underlying methodology is supported by the SemEx tool⁶, which offers two main acquisition functionalities for the selection and normalization of rules. Those functionalities and the corresponding perspectives of SemEx are presented in sections 3.3 and 3.4⁷.

For traceability, diagnosis and revision purposes, the whole set of transformation results needs to be stored and mined. SemEx therefore relies on a rich annotation scheme to encode the rule base under construction and the source text from which it derives. Once it is built, the annotation structure can be explored through dedicated search functionalities. This annotation scheme is described in Section 3.2.

SemEx is built on W3C standards and technologies, which enables the reuse of resources (OWL ontologies) and components (SPARQL search engine).

⁶ <http://www-lipn.univ-paris13.fr/~guisse/index.php?n=Semex.Semex>

⁷ Additional functionalities are accessible *via* SemEx for text annotation and for mining the text and the resulting business model, but they are not described in detail here. See [12] for an overall description of the tool.

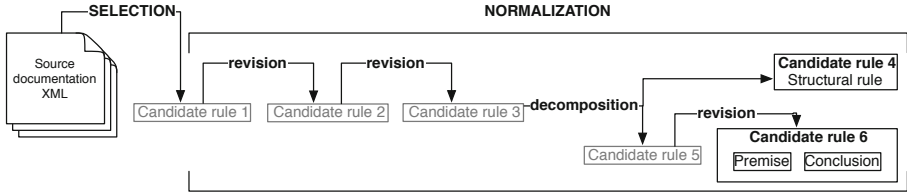


Fig. 2. Rule acquisition overall approach. Two final candidate rules (4 and 6) have been derived from the same initial candidate rule (1) extracted from the source text.

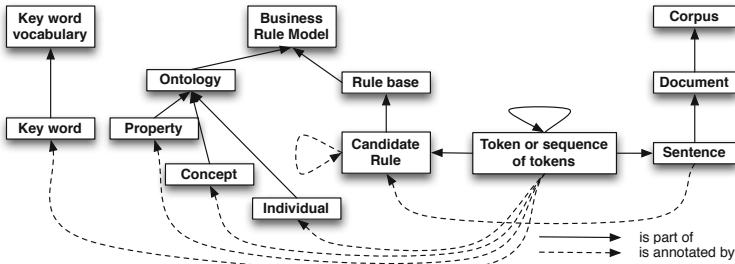


Fig. 3. Annotation scheme

3.2 Annotation Scheme

The annotation scheme describes the data structure in which the source document and the business rules that derive from it are encoded. The basic elements of this scheme are the textual units that are either tokens or sequences of tokens (elementary character strings resulting from a segmentation process). The source document is actually represented as a corpus which is composed of one or several documents. A document is a list of sentences, which are themselves represented as sequences of tokens. The rule base is composed of a set of candidate rules which are also sequences of tokens.

Two types of annotation relations are defined. The high-level ones relate sentences and candidate rules. The rule base is similar to a document but the candidate rules that compose it are partially ordered by an annotation relation. A sentence as a whole can be annotated by a candidate rule which results from its selection and a candidate rule as such can be annotated by one or several candidate rules that derive from it. Low-level annotation relations link the textual units that compose the sentences and the candidate rules to elements of the conceptual (or lexical) and logical (or grammatical) vocabularies: the ontological elements (concepts, properties or individuals) that compose the domain ontology chosen to interpret the source document, and the keywords that serve as grammatical words in the controlled language.

In technical terms, this data structure is encoded as a RDF graph. Annotation links are encoded in RDFa: the RDF annotations are anchored in textual units of XML documents and refer to resources that are OWL entities or candidate rules.

For visualization, low-level annotations are usually represented in a SBVR-style, where the annotated units are colored in blue, red, green and orange according to the type of element (concept, instance, property or known key word) that is referred to. High-level annotations are encoded as explicit references in the source sentence or candidate rule that point to the target candidate rules.

3.3 Selection of Regulatory Fragments

The first challenge for the knowledge engineer who develops a business rule model is to identify the relevant parts of the document and to select the fragments that convey regulatory information. He/she mainly has to read through the source text but the relevant information is often scattered in large and complex documents. At this step, precision must be favored over recall, as it is harder to recover missed fragments than to drop irrelevant ones.

SemEx proposes several devices to help that selection task. The first one is the low level of annotations. When browsing the annotated text in which all the recognized textual units are colored, the expert can focus on passages in which many textual units have been identified. The keywords are especially useful: sentences with a lot of annotated keywords are likely to be relevant.

The second device is a small information extraction engine that allows to design extraction patterns in an interactive mode and look for matching fragments. For instance, a sentence that follows the structure "if... [then]... must..." is likely to convey relevant information. Only a rudimentary version of the extraction engine is integrated in SemEx for now but it could be extended. The goal is to store the patterns so that the most generic and reliable ones can be reused from one application to another.

The third device is a semantic search engine that returns a list of sentences or candidate rules in answer to a semantic query. For instance, if one looks for all the sentences that mention the concept *participant*, one gets all the sentences that contain the word "participant" but also in which the participant is designated as "a participating company" or even as a "member of the program". This ensures a high recall level.

3.4 Normalization

Once the relevant fragment are identified, they must be normalized into CL. The goal is to get rid of ambiguities, to homogenize the lexical and syntactic turns and to make explicit all useful information. This is a complex process that cannot be fully automated. SemEx methodology supports an interactive and progressive process that transforms the initial sentence extracted from the source document into a standardized one, which is written in CL or as close as possible.

In technical terms, the expert can derive a new candidate rule from an existing one. The derivation is encoded as an annotation link and any intermediary step can be restored at any moment. The next section details the transformations that can be applied during the normalization of a rule base. Figure 4 on page 88 gives an example of a derivation tree.

4 Guiding the Transformation Process

This section presents the types of elementary transformations that are needed to translate a NL rule into controlled language. Each of them is explained and illustrated on our use cases. Transformations aim at clarifying the text provided to IT specialists in charge of implementing the rules. This involves reformulating ambiguous or tricky sentences, while preserving the meaning of the underlying rules. Each rule must also be formulated in a self sufficient way, so that its operative meaning can be determined without referring to the source text or to other rules.

Our target language is close to SBVR-SE. The main difference is that we exploit a lexicalized ontology [15] to represent the domain and its conceptual vocabulary. This vocabulary has a narrower scope than that of SBVR, which often includes specific and general-purpose dictionaries⁸ whereas general purpose vocabulary remain mostly out of the scope of our transformations.

We identified four types of transformation that are presented below. The *lexical normalizations* replaces the terms of a candidate rule so as to stick to the domain vocabulary. The *decontextualization* makes explicit the contextual elements of meaning, so that the resulting rules be understandable independently of the source text and other rules. The *syntactic normalization* simplifies the syntactic structure of the sentence so that it is unambiguous and easy to understand. The *semantic normalization* operates at the semantic level, where discourse entities not explicitly referred in the text must often be introduced.

4.1 Lexical Normalization

The operation of lexical normalization aims at checking the business vocabulary of a candidate rule and at replacing all the mentioned terms by their preferred forms. This transformation process takes as input a candidate rule and a lexicalized ontology, which specifies not only the relevant concepts and properties for the field of application but also the preferred and alternative terms to refer to them [15]. The goal is that the candidate rule conform to that vocabulary. The rule terms must be disambiguated and made as specific as possible with respect to the terminology associated to the domain ontology.

The lexical normalization is based on the recognition of the terms of a candidate rule. This operation relies on the annotation of the candidate rule with respect to the ontology: a semantic annotator is integrated in SemEx [12]. Terms recognized as preferred terms in the ontology are left *as is* but alternate terms are replaced by their associated preferred ones. These annotation and replacement processes is based on a lemmatized version of the text as output by a part-of-speech tagger⁹ to ensure the linguistic correctness of the resulting rule.

⁸ In the SBVR-SE specification, vocabularies can use 'Authoritative dictionaries for the relevant natural languages' [14, p.133]. The EU-rent example incorporates Merriam-Webster Unabridged [14, p.275].

⁹ We rely on the TreeTagger

(<http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/>)

The normalization process also requires that the ambiguity of terms which can get several annotations be solved. The expert has to identify the relevant term meaning in the context and select the proper unambiguous preferred terms.

For instance in

*Two **belts** or **restraint systems** are required for the **buckle inspection** and the **low-temperature buckle test**.*

The concepts `BuckleInspection` and `LowTemperatureBuckleTest` are represented by their preferred terms "buckle inspection" and "low-temperature buckle test", but "belts" and "restraint systems", which stand for the concepts `SeatBelt` and `ChildRestraintSystem`, are replaced by the preferred forms of these concepts ("seat belt" and "child restraint system"). This lead to the following transformed candidate rule:

*Two **seat belts** or **child restraint systems** are required for the **buckle inspection** and the **low-temperature buckle test**.*

Lexical normalization also involves nominalizations when a domain concept or entity is mentioned through a verbal phrase. For instance, "be tested for strength" should be replaced by "undergo a strength test" in the following rule:

*All the adjustment devices shall **be tested for strength** as prescribed in paragraph 7.5.1.*

4.2 Decontextualization

Decontextualization extends lexical normalization in that it tracks references to business concepts which are not made by the specific business vocabulary stored in the lexicalized ontology, but by a word or phrase (the referent) which co-refer to a pre-mentioned word or phrase (the antecedent) and whose meaning depends on the antecedent's one. The co-reference link must be broken and the actual meaning of the referent must be made explicit so that the rule can be understood independently of its context. Various types of referent can be found.

Grammatical Words. Pronouns and possessive adjectives often embed a reference to a business entity. In the following rule, "They" should be replaced by "The adjustment devices".

*All the adjustment devices shall undergo a strength test as prescribed in paragraph 7.5.1. **They** must not break or become detached under the tension set up by the prescribed load.*

This type of coreference can often be solved automatically using an anaphora solver, which identifies the referring items and their antecedent. We plan to integrate such an anaphora solver in SemEx and to test the benefit of this additional helping tool but we do not have such an experimental feedback yet.

Generic Business Terms referring to high-level concepts are often used to refer to more specific ones. This is a stylistic way to avoid repetitions when the context is clear enough, but those references must be made explicit in context-independent candidate rules. In the UNO regulation, "Test" is often used for the specific test under description. In the following example, "test" means "micro-slip test" and the generic term must be replaced by the specific one.

*The samples to be submitted to the micro-slip test shall be kept for a minimum of 24 hours in an atmosphere having a temperature of 20 ± 5 °C and a relative humidity of $65 \pm 5\%$. **The test** shall be carried out at a temperature between 15 and 30 °C*

Individual Constraints are often left implicit to skip straightforward details. For instance, in

***Mileage credit** will be credited only to the account of the AAdvantage member who flies, etc.*

"Mileage credit" does not refer to the plain general concept MileageCredit, but to a specific instance earned by the AAdvantage member who took the mentioned flight ("who flies"). Decontextualization yields to

*Mileage credit **awarded for a flight** will be credited only to the account of the AAdvantage member who **takes that flight**, etc.*

Searching for implicit individual constraints is difficult. We plan to compare the concepts in the rule to configuration of roles in the ontology, as the triangle which links the mileage credit earned for a ticket, an AAdvantage member who buys a ticket and a flight for which the ticket is delivered.

Reference Keys are symbols or numbers which refer to a distant piece of text. We observed that in regulation texts, the accompanying text can take various forms but that reference keys are often used to introduce exceptions. Clarification is therefore both important and cumbersome.

In the following example, two load determination procedures must be defined depending on the fact that the buckle is part of the attachment or not. A complex reorganization of the candidate rule is therefore necessary.

(7.5.1) The buckle shall be connected to the tensile-testing apparatus and the load shall then build up to 980 daN . . . If the buckle is part of the attachment, the buckle shall be tested with the attachment, in conformity with paragraph 7.5.2. below,

(7.5.2) The attachments shall be tested in the manner indicated in paragraph 7.5.1., but the load shall be 1,470 daN.

Such a text reorganization cannot be made automatically, but navigation facilities can be proposed so that the expert can easily identify reference keys and get a quick access to the referred parts of text.

4.3 Syntactic Normalization

Syntactic normalization aims at giving a more standard phrasing of the simple candidate rules or at splitting them into several simpler ones that are easier to understand. This normalization is close to the text simplification operations that have been proposed for English [174], but some of these simplifications are not adequate in our business rules context and we propose more specific structures and transformations for our specialized texts.

Sentence Reordering reorganizes the sentence to stick to the order of a logical rule pattern. This often lead to exchange the main and subordinate clauses in a candidate rule. In the following example, the elliptic "upgrade" is understood as a coreference and the clauses are reordered:

*Upgrades are void if sold for cash or other consideration.
If upgrades are sold for cash or other consideration, these upgrades are void.*

Splitting Enumerations. Enumerations are a well-known factor of sentence complexity and splitting enumerations leads to decompose candidate rules into simpler ones. Enumerations have various linguistic forms : pairs of connectors such as *either...or*, *neither...nor*, *not only...but also*, *whether...or*, etc., coordinating conjunctions (*and*, *or*) or plain juxtaposition. The enumerated list can be the subject, the object or even the verb of the clause. For instance, the following candidate rule should be split into three independent sentences:

*Neither accrued mileage, nor award tickets, nor upgrades are transferable by the member upon death.
Accrued mileage is not transferable by the member upon death. Award tickets are not transferable by the member upon death. Upgrades are not transferable by the member upon death.*

Enumerations are difficult to handle automatically. Coordination markers are easy to detect but the scopes of the enumerations are not. Their interpretation is sometimes difficult: splitting is correct only if the enumeration clusters independent conditions; otherwise, it may lead to errors, as in:

Mileage credit may not be combined among AAdvantage members, their estates, successors and assigns.

Splitting Rules. Independently of enumerations, complex candidate rules often need to be split. Solving a coreference is a frequent cause that should be often handled automatically when an anaphora solver will be integrated into Semex. In the following example, the decontextualization of the pronouns "which" and "yours" leads to split the candidate rule into three independent ones:

*The membership year, **which** is the period in **which** your elite benefits are available, runs from March 1 through the last day of February of the following year.*

*The **membership year** is a period. **Member's** elite benefits are available in the **membership year**. The membership year runs from March 1 through the last day of February of the following year.*

4.4 Semantic Restoration

Semantic restoration is a fourth kind of transformation that is often implied by decontextualization or syntactic normalization : discourse entities, which are implicit in the source documents, often have to be restored during normalization.

Restoring an Entity to Solve a Reference. In some cases of decontextualization, there is no unambiguous designation available for solving a coreference and a new entity must be introduced.

In the following example, expliciting "which" by "two perpendicular axes" misses the coreference, using "these perpendicular axes" does not solve it but introducing a reference to a **SensitivityTestAxes** concept enables the rule split.

*When retractors are being tested for sensitivity to vehicle deceleration they shall be tested at the above extraction along two perpendicular axes, **which** are horizontal if the retractor is installed in a vehicle as specified by the safety-belt manufacturer.*

*When retractors are being tested for sensitivity to vehicle deceleration they shall be tested at the above extraction along the **sensitivity test axes**. **Sensitivity test axes** are perpendicular. **Sensitivity test axes** are horizontal if the retractor is installed in a vehicle as specified by the safety-belt. manufacturer.*

Restoring an Interval to Express Constraints. It often happens that constraints between entities are only expressible with the help of an interval that is not mentioned as such in the text.

In the following example, the transformation depends on the concepts available to refer to time entities and the proposed solution assumes there is none and defines them all:

*The breaking load shall be determined within 5 minutes **after** the strap is removed from the conditioning atmosphere or from the receptacle.*

The determination time is the time when the breaking load is determined. The removing time is the time when the strap is removed from the conditioning atmosphere or from the receptacle. The delay between the removing time and the determination time will be less than 5 minutes

These cases frequently occur for temporal and spatial constraints and keyword search (e.g. *after, until, since*) should help to detect the problematic rules.

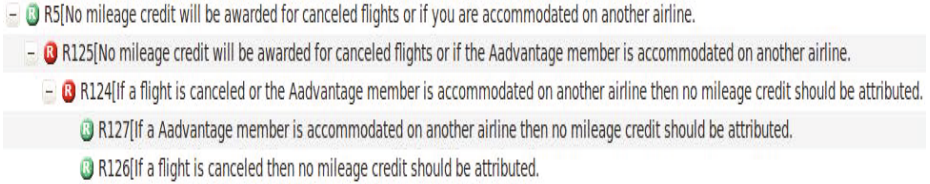


Fig. 4. Example of a derivation tree

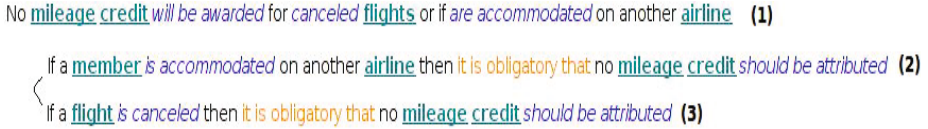


Fig. 5. SBVR translation

5 Experiments and Results

This normalization method has been tested on the two ONTORULE industrial use cases for which two real BR models and rule applications had to be defined, exploiting available sources of information. In each case, a rule base has been designed by extracting a set of candidate rules from the source regulatory texts and by normalizing them, using as many transformation steps as necessary. Transformations which are not yet proposed by a tool incorporated in SemEx have been handled manually by the expert, so as to test the completeness and the correctness of the description.

Figure 4 shows the example of Rule R5, on which a decontextualization transformation (R125) and then a syntactic normalization (R124) are applied before it is decomposed into two elementary and independent sub-rules (R126 and R127). We tried to automatically translate the initial and the final candidate rules into SBVR, using the NL2SBVR tool [2]. The result (Figure 5) shows that the meaning of the initial sentence is lost in the translation (statement 1), but that the translation works for the transformed final candidate rules giving valid SBVR statements (2 and 3).

The following tables present the types and size of the rule bases that have been designed out of the source regulatory texts for each use case. Four types of transformations are considered here: the normalization of the vocabulary; the restoration of contextual information, the syntactic transformation and the decomposition of rules. Since the syntactic and semantic transformations are often performed together, the semantic transformations are not isolated here.

Table 1 shows the results of the selection phase. It gives the size of the initial rule bases with respect to the size of the source texts. In the use cases, 1/4 at least of the sentences have been selected as relevant regulatory information.

This high rates are due to the fact that the source text are short but dense. This table also shows the initial structure of the rule base. More than 2/3 of the extracted rule are structural rules, some of them are operative rules and few of them are derivation rules.

Table 1. Results of the selection phase on the AAdvantage and Audi use cases (CR = candidate rule; SR = structural rule; OR = operative rule; DR = derivation rule)

Use Case	# of sentences	# of initial CR	Selection rate	# of SR	# of OR	# of DR
AAdvantage	245	74	30%	54	14	6
Audi	221	54	25%	45	9	0

Table 2 details the types of transformations that have been made for each use case. Four types of transformation are considered: the first three ones respectively affect the vocabulary, the context and the syntax; the last one is the decomposition of one candidate rule into several ones. In terms of number of transformations, the syntactic normalization is the most expensive one: it requires twice as much transformation steps than other types of transformations. All the initial rules have undergone a syntactic normalization (100%). The vocabulary transformation and decontextualization also affect more than half of the initial candidate rules, whereas decomposition is required in less numerous cases (resp. 35% and 20% for the AAdvantage and Audi use cases).

Table 2. Distribution of the different types of transformation (α = rate of a given type of transformation when considering all the transformations that have been made; β = rate of initial candidate rules that have undergone a given type of transformation)

Normalization types	AAdvantage α	AAdvantage β	Audi α	Audi β
Vocabulary	19%	65%	20%	61%
Context	18%	60%	19%	57%
Syntax	43%	100%	47%	100%
Decomposition	21%	35%	14%	20%

The last table 3 presents the structure of the resulting rule base, the number and types of the final candidate rules. As expected, there are more final rules than initial ones since some of them have undergone a decomposition. At the end of the normalization process, the regulatory information written in the AAdvantage text (245 sentences) has been reduced to a set of 104 candidate rules which are autonomous SBVR statements. The reduction is even higher for the Audi use case (225 sentences, 65 candidate rules), whose source text is more detailed.

Table 3. Structure of the final rule base

Use Case	SR	OR	DR	Final CR
AAdvantage	71	27	6	104
Audi	54	11	0	65

6 Conclusion

This paper tackles the rule acquisition problem, assuming that regulations written in NL are a rich source of knowledge but that turning NL into formal statements is a complex task than cannot be fully automated. We propose to decompose the acquisition process into two main phases: the translation of NL statements into CL and their formalization into an operational rule base.

The present paper focuses on the first "normalization" phase. It shows that transforming NL statements into CL is itself a complex task that involves some lexical and syntactic normalizations but also the restoration of contextual information and of implicit semantic entities to get a set of self-sufficient, unambiguous and easy to understand rule statement. We also present the SemEx tool that supports the proposed acquisition methodology based on the selection of the relevant text fragments and their normalization into a SBVR-like CL.

SemEx has been designed as an interactive rule acquisition tool. It guides the domain expert through a sequence of steps that produces elementary candidate rules, helps the detection of relevant keywords and controls the results. Some helping tools have already been plugged in SemEx: *e.g.* a semantic annotator that takes a lexicalized ontology as input and annotates a text with respect to that ontology and a keyword search that helps locating the most relevant text fragments or identifying the problematic features (*e.g.* anaphoric pronouns) in the selected fragments.

In the next future, we plan to exploit more intensively NL processing tools to guide and help the acquisition task. Part of the morphological and syntactic calculus could be automated using a parser. Some anaphora could be solved and syntactic transformation patterns could be exploited. We are currently testing these technologies to enrich SemEx with additional helping tools. We are also planning to integrate a SBVR parser to check the syntactic conformity of the final candidate rules with SBVR-SE. Successful checking would indicate that the transformation phase is achieved and failures would give some indication on how to complete it. The last and most challenging tool would be a semantic parser able to check the conformity of the final candidate rule to the underlying ontology. That would help to identify the semantic shortcuts that need to be made explicit in the candidate rules.

References

1. Bajec, M., Krisper, M.: Issues and challenges in business rule-based information systems development. In: ECIS (2005)
2. Bajwa, I.S., Lee, M.G., Bordbar, B.: Sbrv business rules generation from natural language specification. In: AAAI Spring Symposium 2011 Artificial Intelligence 4 Business Agility, pp. 541–545. AAAI Press, San Francisco (2011)

3. BRG: Defining business rules what are they really? The Business Rules Group : formerly, known as the GUIDE Business Rules Project - Final Report revision 1.3 (July 2000)
4. Brodie, C., Karat, C.-M., Karat, J.: An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In: SOUPS 2006 (2006)
5. Candido Jr., A., Maziero, E., Gasperin, C., Pardo, T.A.S., Specia, L., Aluisio, S.M.: Supporting the adaptation of texts for poor literacy readers: a text simplification editor for brazilian portuguese. In: Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications, EdAppsNLP 2009, pp. 34–42. Association for Computational Linguistics, Stroudsburg (2009)
6. Chandrasekar, R., Doran, C., Srinivas, B.: Motivations and methods for text simplification. In: Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING 1996), pp. 1041–1044 (1996)
7. Chandrasekar, R., Srinivas, B.: Automatic induction of rules for text simplification (1997)
8. Dinesh, N., Joshi, A., Lee, I., Sokolsky, O.: Reasoning about Conditions and Exceptions to Laws in Regulatory Conformance Checking. In: van der Meyden, R., van der Torre, L. (eds.) DEON 2008. LNCS (LNAI), vol. 5076, pp. 110–124. Springer, Heidelberg (2008)
9. Dubauskaite, R., Vasilecas, O.: An open issues in business rules based information system development. In: Innovative Infotechnologies for Science, Business and Education, vol. 1 (2009)
10. Gasperin, C., Specia, L., Pereira, T.F., Aluisio, S.M.: Learning when to simplify sentences for natural text simplification. In: ENIA 2009 (VII Encontro Nacional de Inteligência Artificial) (2009)
11. Halle, B., Goldberg, L., Zackman, J.: Business Rule Revolution: Running Business the Right Way. Happy About (2006),
<http://books.google.com/books?id=I3mvAAAACAAJ>
12. Lévy, F., Nazarenko, A., Guissé, A., Omrane, N., Szulman, S.: An environment for the joint management of written policies and business rules. In: Proceedings of the International Conference on Tools with Artificial Intelligence (IEEE-ICTAI 2010), pp. 142–149 (2010)
13. Max, A.: Simplification interactive pour la production de textes adaptés aux personnes souffrant de troubles de la compréhension. In: Proceedings of TALN, poster session (2005)
14. OMG: Sbr (2008), <http://www.omg.org/spec/SBVR/Current>
15. Omrane, N., Nazarenko, A., Rosina, P., Szulman, S., Westphal, C.: Lexicalized ontology for a business rules management platform: An automotive use case. In: Proceedings of the 5th International Symposium on Rules, International Business Rules Forum (RuleMF@BRF), Ft Lauderdale, Florida, USA (November 2011)
16. Ross, R.G.: Principles of the Business Rule Approach, ch. 8-12. Addison-Wesley, Boston (2003)
17. Siddharthan, A., Caius, G.: Syntactic simplification and text cohesion (2003)
18. Wagner, G., Lukichev, S., Fuchs, N.E., Spreeuwenberg, S.: First-version controlled english rule language. In: REVERSE IST 506779 Report I1-D2 (February 2005)

A Rule Based Approach for Business Rule Generation from Business Process Models

Saleem Malik¹ and Imran Sarwar Bajwa²

¹ Department of Computer Science & IT,
The Islamia University of Bahawalpur, Pakistan

² School of Computer Science, University of Birmingham, UK
saleemmalik35@yahoo.com, i.s.bajwa@cs.bham.ac.uk

Abstract. In this paper, a rule based approach is presented to translate Business Process Model Notation (BPMN) based business process models into Semantics of Business Vocabulary based Rules (SBVR) based business rules. Such translation can simplify the process of understanding the information represented in BPMN models for the business stakeholders as information represented in business rules is easy to understand instead of a BPMN based graphical representation of a business process model. In this paper, we also present a case study to validate the performance of the case study.

Keywords: Business Process Modelling, BPMN, SBVR.

1 Introduction

In typical Business Process Modeling [1] (BPM), business analysts analyze business information and model into business processes. To attain, efficient and quality business processes, a standard Business Process Modelling Notation [2] (BPMN) is used. BPMN supports graphical representation of a business process model. Once, a business process model is ready, it is demonstrated to the external business stakeholders to validate the correctness of the information represented in the model. However, it is a common knowledge that a graphical representation of a business process model can be complex to understand for the business stakeholders. While, a natural language (NL) representation of a BPMN based model can be easy to understand for external business stakeholders. In this paper, to address the above discussed challenge, we present a novel approach to translate a BPMN-based business process model to a NL representation such as SBVR [3] business rules. In BPMN to SBVR translation, SBVR is used a pivot representation and simplifies the translation process.

The rest of the paper is structured as follows. Section 2 presents background and related work of the presented research. In Section 3, we present a framework used to generate SBVR based natural language expressions from the BPMN based graphical business process models. Section 4 presents a case study, finally, paper is concluded with the future work.

2 Background and Related Work

2.1 Business Process Modelling Notation (BPMN)

BPMN defines a Business Process Diagram (BPD), which is based on a flowcharting technique tailored for creating graphical models of business process operations. It is a notation that is understandable by all business users: from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology. A BPMN model consists of simple diagrams with a small set of graphical elements.

Flow Objects: The flow objects [2] are used to define the behavior of a business process. Three flow objects are commonly used such as Activities, Events, and Gateways.

Flow Object Connectors: The flow objects can be connected to each other by using following three connectors such as Sequence flows, Message flows, and Associations.

Swimlanes: The swimlanes or partitions are used to group various objects involved in a process. Two common types of swimlanes are Pools and Lanes.

Artifacts: In a business process model, artifacts can be used to represent additional information related to the process. Commonly used artifacts in BPMN models are Data Objects, Groups and Annotations.

2.2 Semantic Business Vocabulary and Rules (SBVR)

Semantics of Business Vocabulary and Rules [3] is an adopted standard used to specify the business rules.

SBVR Business Vocabulary. A business vocabulary [3] (section: 8.1) consists of all the specific terms and definitions of concepts used by an organization or community in course of business. In SBVR, there are four key elements:

- An *Object Type* is a general concept e.g. Employee etc.
- An *Individual Concept* is a qualified noun e.g. 'Birmingham', a famous city.
- A *Characteristic* is an abstraction of a property of an object e.g. name of city.
- A *Verb Phrase* is a verb in English sentences e.g. employee uses account.

SBVR Business Rules. A SBVR business rule is a formal representation 'Under business jurisdiction' [3]. Each SBVR business rule is based on at least one fact type. The SBVR rules can be a structural rule [ibid] used to define an organization's setup or a behavioural rule [ibid] used to express the conduct of a business entity.

2.3 Related Work

SBVR can be used as an intermediate representation in translation of formal representations to natural language and vice versa. Examples of such translations are UML/OCL to SBVR [5], SBVR to UML models [6], SBVR to OCL constraints [7, 9], etc. A SBVR based representation contains a set of business vocabulary and business rules. As far as we know, the presented approach is the first proposal to provide such translation. Moreover, our approach provides a standard format (such as SBVR) for defining the business rules in natural languages.

3 Translating BPMN to English

In this section we explain the mapping of all key elements in BPMN to their respective elements in SBVR metamodel to extract SBVR vocabulary. Then the SBVR vocabulary is mapped to Business rules and the Business rules are also represented using SBVR Structured English notation to make it easy to read. Details of the used framework for BPMN to SBVR translation is given below.

3.1 Input BPMN Model

To generate SBVR based English translation of a business process model, a XML or XMI representation of a BPMN model is used. The XML representation of a BPMN model can be used any CASE tool as most of the CASE tool provides this facility. However, we have used the Enterprise Architect tool [10] to generate a BPMN model and we exported the XML representation of the same BPMN model by using the Enterprise Architect tool.

3.2 Mapping Flow Objects

Flow objects are the main describing elements within BPMN, and consist of three core elements: events, activities, and gateways. Mapping of all three elements is presented below:

Mapping Events: Start event is mapped to the initiation of the SBVR specification and End event is mapped to the end of the SBVR specification. To handle Start event, we add a string “The process of” + Model Name + “starts with” to text of the Start event. Here, XML file name is used as Model Name. However, to handle End event, a string “The process of” + Model name + “ends with” is added to the End event text.

There can be some other types of Events such as Throwing (use to represent a completion message when a process ends) event or Catching (used to represent an incoming message starts a process) event. However, current implementation only supports Start event and End event.

Mapping Activity: An Activity is mapped to a Logical Formulation to be used in the *consequent* part of an implication or in Necessity Formulation. Following can be two possible cases for mapping Activity:

- An activity is mapped to the Atomic Formulation in an *ActivityFactType*. In Activity to *ActivityFactType* mapping, the Object Type referred by role1 and the name of the Object Type referred to by role2 is mapped to the actor of the Activity.
- An Activity without any condition is potentially an initial Activity and an initial activity is mapped to a Necessity Formulation in a Logical Formulation of SBVR model and a Necessity Formulation is represented by keywords “It is necessary”.

Gateway: In a BPMN model, a Gateway represents the conditions such as OR, AND, etc. An example of BPMN gateway to English mapping is shown in Table 1.

Table 1. Mapping BPMN Gateways to SBVR Logical Formulation

BPMN Element	SBVR Element
XOR	Exclusive Disjunction
OR	Disjunction
AND	Conjunction

3.3 Mapping Flow Object Connectors

Flow objects are connected to each other using Connecting objects, which are of three types: sequences, messages, and associations. We translate the only connecting objects with captions as shown in Table 2:

Table 2. Mapping BPMN Flow Objects to SBVR representation

BPMN Element	SBVR Element
Sequence Flow	Activity-A results in Activity-B
Conditional Flow	Activity-B results if condition is True
Message Flow	Pool-A connected with Pool-B
Association	Fact Type (Artifact/Text is connected to Flow Object)

3.4 Mapping Artifacts

BPMN was designed to allow modelers and modeling tools some flexibility in extending the basic notation and in providing the ability to additional context appropriate to a specific modeling situation, such as for a vertical market (e.g., insurance or banking). Any number of Artifacts can be added to a diagram as appropriate for the context of the business processes being modeled. The current version of the BPMN specification pre-defines only three types of BPD Artifacts, which are shown in Table 3:

Table 3. Mapping BPMN artefacts to SBVR representation

BPMN Element	SBVR Element
Data Objects	Source of a information
Group	Activities in a Group are written together.
Annotation	Additional Information

3.5 Mapping Swim Lanes

As Swim-lanes are use to group activities in a business process model, similarly, we have used Pool and Lane to group the business rules. All activities shown in a single Pool or Lane are grouped together.

3.6 Process Dependencies

The process dependencies of the BPMN model are mapped to the Logical-Formulations of the SBVR model. For this transformation, we identified two alternative mappings, which depend on where the Logical Formulation is defined: (1) in the condition part of implications, or (2) in the consequent part of implications or in a Necessity Formulation.

Mapping Relation: A Relation is mapped to a Logical Formulation used in the *condition* part such as an ‘And’ Relation is mapped to conjunction and a Disc Relation is mapped to disjunction.

Mapping Variables: A variable is mapped to an Atomic Formulation in a *Unary-Fact Type*, *Association-Fact Type* or *IsOfPropertyFactType*. Here, a variable can be of any type; either the Variable updated by the Activity or a Precondition of the Activity.

Mapping Literals: A literal is mapped to a simple Atomic Formulation that is not based on a Fact Type and has only one binding Noun Concept, where the Noun Concept will be represented as an Individual Concept.

3.7 Optimizing the English Representation

In table 1, 2, and 3, we have shown the way various BPMN elements are mapped to SBVR based English. However, English generated in these examples is difficult to understand. Hence we need to optimize the generated English to make it easy to read and make it understandable. For the sake of optimization we have performed following two steps:

Resolving Phrases: In this phase the unstructured phrases extracted from BPMN elements is structured to make the extracted information sensible. To restructure the phrases following steps were performed:

Process Activity Text: The text in Activity symbol is processed as we append the text “user” at the start of the Activity symbol text. For example, the text “Buy item now” is processed as “User buys item now”. Here, we do add ‘s’ with the verb to keep grammar correct. We have used WordNet [8] version 3.1 to identify possible POS tags for each token of the text.

Process Gateway Text: The text in a gateway is handled in various ways.

- i. If the Gateway poses a Yes/No question then two copies of the text are generated: one copy with positive sense and second copy with the negative sense. To generate a positive sense we simple add a helping verb in between Noun and Verb. While, for generating the copy with negative sense we also add token “not” with the helping verb. For example the text “Item Sold” is structured to “It is sold” and “Item is not sold”.
- ii. If the Gateway does not pose Yes/No question then we generate two copies of the Gateway text with by adding the text of respective branch. For example, the text “auction type” is optimized to “auction type is buy now” and “auction type is bid for item”.

Applying Structured English Notation: Finally, we apply SBVR structured English notation to generated SBVR rule. Here, common nouns or Object Type are underlined e.g. employee; the verbs are represented as Verb Concept and are italicized e.g. *uses*; the SBVR keywords are bolded e.g. **It is obligatory**; the proper nouns are represented as Individual Concepts by double underlining e.g. London.

4 A Case Study

To demonstrate the potential of the presented approach, a small case study is discussed from the domain of Item Sale system (see Figure 1) that is online available in BPMN tutorial by IBM [11]. Following is the problem statement of the case study:

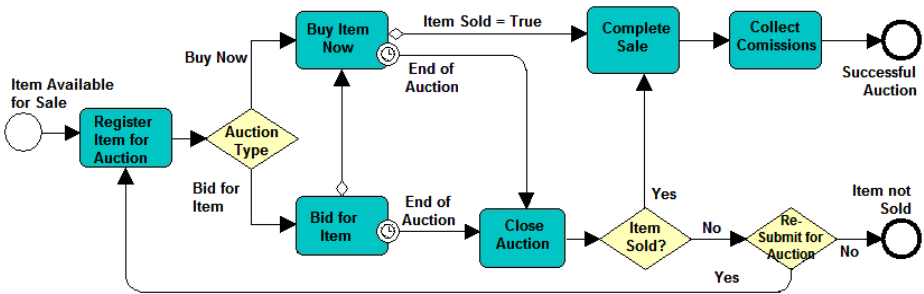


Fig. 1. Item Sale Process Model [11]

The problem statement of the case study was given as input (XML representation) to the BR-Generator tool. The tool parses the XML representation tags and extracts the SBVR vocabulary by performing BPMN to SBVR mapping. The complete mapping from BPMN to SBVR for solved case study is shown in Table 4:

Table 4. SBVR vocabulary generated from BPMN XML representation

<i>Details</i>
<p>The Sale Item process <i>starts</i> with <u>item is</u> available for <u>sale</u>. It is necessary that the <u>user registers item</u> for <u>auction</u>.</p> <p>If <u>auction type is buy</u> now then it is necessary that <u>user buys item</u> now. When there <i>is</i> end of <u>auction</u>, it is necessary that <u>user closes auction</u>. If <u>Item is sold</u> then it is necessary that <u>user completes sale</u>. It is necessary that <u>user collects commissions</u>.</p> <p>The Sale Item process <i>ends</i> with successful <u>auction</u>.</p> <p>If <u>auction type is bid</u> for <u>item</u> then It is necessary that <u>user bids</u> for <u>item</u>. When there <i>is</i> end of <u>auction</u>, it is necessary that <u>user closes auction</u>. If <u>item is sold</u> then it is necessary that user <u>completes sale</u>. If <u>item is</u> not <u>sold</u> them <u>user re-submits</u> for <u>auction</u>.</p> <p>If <u>user re-submits</u> for <u>auction</u> then it is necessary that <u>user registers</u> item for <u>auction</u>. If <u>user not re-submit</u> for <u>auction</u> then Sale Item process <i>ends</i> with <u>item is</u> not <u>sold</u>.</p>

There are few limitations of the approach e.g. “When there *is* end of auction, **it is necessary** that user close auction.” Moreover, there are a few grammatical mistakes such as “If auction type is buy now” and “If user not re-submit for auction”. We plan to provide grammar correction facility in the future research.

4.1 Evaluation

We have done performance evaluation to evaluate that how accurately the BPMN model notation is translated to SBVR based English specification by our tool BR-Generator. There are total 26 BPMN symbols of 4 types in the solved case study problem those were translated to SBVR 1.0 based English sentences. In Table 5, the average recall for SBVR software requirement specification is calculated 88.46% while average precision is calculated 92.00%.

Table 5. Results of BPMN to SBVR based English Translation

<i>Type/Metrics</i>	N_{sample}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	<i>Rec%</i>	<i>Prec%</i>
Software Requirements	26	23	2	1	88.46	92.00

Table 6. Usability Survey Results

<i>User</i>	<i>Easy to do</i>		<i>Correct Understanding</i>	
	Manual	By Tool	Manual	By Tool
Novice	30%	90%	36%	87%
Medium	55%	85%	72%	82%
Average	42.50%	87.50%	54.00%	84.50%

Besides measuring accuracy we also conducted a survey to measure the effectiveness of the presented approach. We made two groups with 10 members in each group. First, we gave them three BPMN process models to interpret. Then we told them to interpret those three BPMN models using our tool BR-Generator. Then we gave 1 to 10 score under easy to do and correct understanding categories. Though the accuracy of the tool is a bit concern but we can overcome this in future work by improving the implementation. The average results we received are shown in Table 6:

5 Conclusion and Future Work

In this paper, we presented a rule based approach that can be helpful in understanding the complex BPMN models specifically for the novel users that can lead to a better feedback from the Business stakeholders ultimately resulting in better business process models those are more acceptable for Business analysts and Business stakeholders. Moreover, the SBVR based output generated by the tool can be used for automated transformation to other formal specifications such as BPEL, UML, OCL, etc. Additionally, the BPMN models can be analyzed for consistency by translating the output of our approach (such as SBVR) to Alloy that globally accepted language used for model analysis.

References

- [1] Korherr, B.: Business Process Modelling, VDM Verlag Saarbrücken, Germany (2008) ISBN: 3836487160 9783836487160
- [2] Object Management Group. Business Process Definition Metamodel. Version 1.0.2 (2004), <http://www.bpmn.org/Documents/BPDM/OMG-BPD-2004-01-12-Revision.pdf>
- [3] Business Process Management Initiative: Business Process Modeling Notation. Specification Version 1.0 (May 3, 2004), <http://www.bpmn.org/>
- [4] Object Management Group. Semantics of Business vocabulary and Rules. (SBVR) Standard v.1.0. Object Management Group (2008), <http://www.omg.org/spec/SBVR/1.0/>
- [5] Pau, R., Cabot, J.: Paraphrasing OCL Expressions with SBVR. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds.) NLDB 2008. LNCS, vol. 5039, pp. 311–316. Springer, Heidelberg (2008)
- [6] Raj, A., Prabhakar, T.V., Hendryx, S.: Transformation of SBVR business design to UML models. In: Proceedings of the 1st India Software Engineering Conference, February 19-22 (2008)
- [7] Bajwa, I.S., Bordbar, B., Lee, M.G.: OCL Constraints Generation from Natural Language Specification. In: 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2010), pp. 204–213 (2010)
- [8] WordNet: An Electronic Lexical Database. MIT Press, <http://wordnet.princeton.edu/wordnet/publications/>
- [9] Bajwa, I.S., Naeem, M.A., Ali, A., Ali, S.: A Controlled Natural Language Interface to Class Models. In: 13th International Conference on Enterprise Information Systems (ICEIS 2011) pp.102–110 (2011)
- [10] Sparx Systems, Enterprise Architect, <http://www.sparxsystems.com.au/>

Reaction RuleML 1.0: Standardized Semantic Reaction Rules

Adrian Paschke¹, Harold Boley², Zhili Zhao¹,
Kia Teymourian¹, and Tara Athan³

¹ Freie Universitaet Berlin, Germany
{paschke,zhili,teymourian}@inf.fu-berlin.de

² Information and Communications Technologies,
National Research Council Canada
Fredericton, NB, Canada

harold.boleynrc.gc.ca

³ Athan Services, W Lafayette, IN, USA
taraathan@gmail.com

Abstract. RuleML is a family of XML languages whose modular system of schemas permits high-precision (Web) rule interchange. The family's top-level distinction is deliberation rules vs. reaction rules. In this paper we address the Reaction RuleML subfamily of RuleML and survey related work. Reaction RuleML is a standardized rule markup/serialization language and semantic interchange format for reaction rules and rule-based event processing. Reaction rules include distributed Complex Event Processing (CEP), Knowledge Representation (KR) calculi, as well as Event-Condition-Action (ECA) rules, Production (CA) rules, and Trigger (EA) rules. Reaction RuleML 1.0 incorporates this reactive spectrum of rules into RuleML employing a system of step-wise extensions of the Deliberation RuleML 1.0 foundation.

1 Introduction

Event-driven reactive functionalities are urgently needed in present-day distributed systems and dynamic Web-based environments. Reaction rules constitute a promising approach to specify and program such reactive systems in a declarative manner. In particular, they provide the ability to reason over events, actions and their effects, and allow detecting events and responding to them automatically. A great variety of approaches have been developed for reaction rules, which have for the most part evolved separately and have defined their own domain and platform specific languages [14, 12, 16]. Novel semantics are being devised, including for the Logic-based agent and Production System language (LPS) and KELPS [6].

Reaction RuleML [7] is intended as a common standard for representing reaction rules and rule-based complex event processing (CEP) in a platform independent

¹ <http://reaction.ruleml.org/>

XML markup expression language. Reaction RuleML allows for standardized rule interchange, semantic interpretation and translation, and distributed event-messaging interactions in loosely-coupled and de-coupled distributed rule-based systems such as Web inference services and semantic agents.

RuleML² has been designed for the standardized interchange of the major kinds of rules in an XML format that is uniform across rule languages and platforms. It has broad coverage and is defined as an extensible family of languages. In this paper, we introduce Reaction RuleML 1.0³, which directly builds on RuleML 1.0^{3,4} By describing the language features of Reaction RuleML this paper also surveys the major lines of reaction rule types. We assume that readers are already familiar with Web rule technologies and their underlying semantics⁵

The rest of the paper is organized as follows. Section 2 introduces the main rule syntax of Reaction RuleML 1.0 and specializes it to the four subbranches addressing the major reaction rule types. Section 3 describes selected expressive features of Reaction RuleML. In section 4 we compare Reaction RuleML with other platform-independent rule standards, analyzing its representational completeness with respect to an ontological reference metamodel. Section 5 summarizes the approach of Reaction RuleML 1.0, discusses its applicability with respect to recent works, and gives an outlook on future work in Reaction RuleML 1.1.

2 Reaction RuleML for Representing Reaction Rules

Reaction rules are concerned with the invocation of actions in response to events and actionable situations¹². They state the conditions under which actions must be taken and describe the effects of action executions. In the last decades various reaction rule languages and rule-based event processing approaches have been developed, which for the most part have been advanced separately^{14, 16}.

Reaction RuleML follows the general principles of markup language design as defined in¹¹. Its subbranches span across the four major reaction rule types:

- Production Rules (Condition-Action rules) in the Production RuleML branch
- Event-Condition-Action (ECA) rules in the ECA RuleML branch
- Rule-based Complex Event Processing (CEP) (complex event processing reaction rules, (distributed) event messaging reaction rules, query reaction rules etc.) in the CEP RuleML branch
- Knowledge Representation (KR) Event/Action/Situation Transition/Process Logics and Calculi in the KR Reaction RuleML branch

Reaction rules are defined by a general `<Rule>` element which can be specialized in the different Reaction RuleML branches to the four major types of reaction

² <http://ruleml.org/>

³ <http://ruleml.org/reaction/1.0/>

⁴ <http://ruleml.org/1.0/>

⁵ For a deeper study of Web rule / event processing technologies we refer to^{10-12, 7}.

rules (and variants thereof). The following template shows the most general rule syntax of RuleML with a focus on Reaction RuleML. We use 1- or 2-letter indicators for syntax from Deliberation (D), Reaction (R), or Deliberation+Reaction (DR) RuleML.

```
<Rule @key @keyref @style>
  <!-- rule info and life cycle management, modularization -->
    <meta> <!-- DR: (semantic) metadata of the rule --> </meta>
    <scope> <!-- R: scope of the rule e.g. a rule module --> </scope>

  <!-- rule interface description -->
    <evaluation> <!-- R: intended semantic profiles --> </evaluation>
    <signature> <!-- R: rule interface signature and modes --> </signature>

  <!-- rule implementation -->
    <qualification> <!-- R: e.g. qualifying rule declarations, e.g.
      priorities, validity, strategy --> </qualification>
    <quantification> <!-- DR: quantifying rule declarations,
      e.g. variable bindings --> </quantification>

    <on> <!-- R: event part --> </on>
    <if> <!-- DR: condition part --> </if>
    <then> <!-- D: (logical) conclusion part --> </then>
    <do> <!-- R: action part --> </do>
    <after> <!-- R: postcondition part after action,
      e.g. to check effects of execution --> </after>
    <else> <!-- DR: (logical) else conclusion --> </else>
    <elsedo> <!-- R: alternative/else action,
      e.g. for default, exception handling --> </elsedo>
</Rule>
```

These role tag elements below the general `<Rule>` element are used for representing the following information blocks in a rule.

- The attribute `@keyref` is used for creating distributed and modularized accessibility within a (distributed) knowledge base, where `@key` is the identifier key and `@keyref` is a key reference.
- The general style of a reaction rule is defined by the optional attribute `@style`, which has the following values in Reaction RuleML
 - **active**: actively polls and detects occurred events in ECA and CEP rules or changed conditions in production rules.
 - **messaging**: waits for incoming complex event message (inbound) and sends messages (outbound) as actions.
 - **reasoning**: logical reasoning as e.g. in formalisms such as event / action / transition logics (as e.g. in Event Calculus, Situation Calculus, temporal action languages formalizations) etc.
- The metadata `<meta>` is used to annotate the rule with optional metadata.
- The scope `<scope>` defines a (constructive) view on the rulebase, e.g. the rule only applies to a particular module in the rulebase.
- The evaluation semantics (interpretation semantics and/or execution semantics) of reaction rules is defined in the optional role subchild `evaluation`. This can be used to define rule evaluation semantics such as weak or strong

evaluation which defines the “execution lifecycle” of the rule execution or other semantic policies, e.g. event consumption policies, transaction semantics etc.

- The `<signature>` defines the rule signature with optional input / output mode declarations. The rule signature declaration can act as public rule interface and can be published together with the intended evaluation semantics in distributed Reaction RuleML bases.
- The qualification `<qualification>` defines an optional set of rule qualifications such as a validity value, fuzzy value or a priority value.
- The quantification `<quantification>` is used to define quantifiers such as the typical existential and universal quantification; it can also be used for extensions such as variable binding patterns to restrict pattern matching in production rules or define other operator definitions.
- The `<on>` part of a rule defines the triggering events, which can be atomic or complex event definitions.
- The `<if>` part defines one or more conditions. In case of a reaction rule with actions this would be the pre-conditions.
- The `<then>` part defines the (logical) conclusions of the rule. Hybrid rules are possible which define a logical conclusion in the `<then>` part as well as actions in the `<do>` part.
- The `<do>` part of a rule defines the actions, which can be an atomic as well as a complex action definition.
- The `<after>` part of a rule defines the post-conditions which hold after the actions. They can be used, e.g., as post-conditional integrity constraints on the effects of the action execution on the knowledge state, which holds after the action has been performed. Depending on the defined `<evaluation>` semantics this might lead to roll-backs in the case of transactional semantics.
- The `<else>` part leads to if-then-else rules. Special semantics for `<else>` might be defined in the `<evaluation>` semantics of the rule.
- The `<elsedo>` part is executed instead of the `<do>` part. This allows to define e.g. compensating actions in transactional logics or default and exception handling actions if the normal execution of the `<do>` fails. Special semantics for `<elsedo>` might be defined in the `<evaluation>` semantics of the rule.

Depending on which parts of this general rule syntax are used, different types of reaction rules can be expressed, e.g. if-then (derivation rules, as used e.g. in KR RuleML for logical event/action calculi), if-do (production rules), on-do (trigger rules), on-if-do (ECA rules).

Derivation Rule:	Production Rule:	ECA Rule:	CEP Rule:
<code><Rule style="reasoning"></code>	<code><Rule style="active"></code>	<code><Rule style="active"></code>	<code><Rule style="messaging"></code>
<code><if>...</if></code>	<code><if>...</if></code>	<code><on> ---- </on></code>	<code><on> event 1 </on></code>
<code><then>---</then></code>	<code><do>---</do></code>	<code><if> ... </if></code>	<code><do> action 1 </do></code>
<code></Rule></code>	<code></Rule></code>	<code><do> ---- </do></code>	<code><on> event 2 </on></code>
		<code></Rule></code>	<code><if> condition</if></code>
			<code><do> action 2 </do></code>
			<code>...</code>
			<code></Rule></code>

Before we further describe the expressive features in Reaction RuleML 1.0 in section 3, we first specialize this general reaction rules approach to the four different Reaction RuleML branches in the following subsections.

2.1 Production RuleML

A production rule is a statement of rule programming logic, which specifies the execution of one or more actions in case its conditions are satisfied, i.e. production rules react to states changes (not to explicit events as e.g. in ECA rules). The essential syntax is *if Condition do Action* as shown in the following example (this is a ‘pure’ production rule since the `<do><Assert><Atom>` action in Reaction RuleML maps to a `<then><Atom>` conclusion in Deliberation RuleML):

```
<!-- If premium customer and regular product do assert discount of 5 percent
      for customer on product -->
<Rule style="active">
  <if>
    <And>
      <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
      <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
    </And>
  </if>
  <do>
    <Assert>
      <Atom><Rel>discount</Rel><Data>5.0</Data><Var>cust</Var><Var>prod</Var></Atom>
    </Assert>
  </do>
</Rule>
```

The central predefined actions in Production RuleML are: `<Assert>` (add knowledge); `<Retract>` (retract knowledge); `<Update>` (update/modify knowledge); `<Set>` and `<Get>` (assignment/dereferencing of (global) variables); `<Execute>` (execution of (external) functions). Furthermore, a generic `<Action>` is defined which allows model references to externally defined action models, e.g., `<Assert>` is a shortcut for `<Action type="ruleml:Assert">`, where the action such as for assertions is defined in the metamodel of Reaction RuleML (i.e. in the Reaction RuleML meta-ontology).

Production RuleML supports “Negation-as-failure” (`<Naf>`), which by default is interpreted with an inflationary semantics⁶. Again, `<Naf>` is a shortcut for the more generic negation definition `<Negation type="ruleml:InflationaryNegation">` with reference to the Production RuleML metamodel. Other types of negations can be specified using the `@type` attribute.

In the `evaluation` section the semantics for the interpretation and operational execution can be specified. The Reaction RuleML metamodel predefines typical semantic profiles (see section 3.2) for different classes of production rule systems. This furthermore includes semantics for conflict resolution strategies such as `ruleml:Refraction`, `ruleml:Priority`, and `ruleml:Recency`.

⁶ For a discussion in RIF-PRD see <http://lists.w3.org/Archives/Public/public-rif-wg/2008Dec/0053.html> and <http://lists.w3.org/Archives/Public/public-rif-wg/2008Dec/0055.html>

The rule quantification can be used to quantify variables and define further binding patterns. For instance, the following example defines a rule with a `<Forall>` quantifier for one variable with a variable binding pattern. In contrast to Deliberation RuleML 1.0 the `<quantification>` is defined under the `<Rule>` and not outside of Deliberation RuleML's rule element `<Implies>`. The reason for this is, because Reaction RuleML defines both the interface and the implementation under the `<Rule>`, but possibly distributed within the KB with a reference from the `<Rule>`'s interface description to the `<Rule>`'s implementation in which the quantifier are defined.

```
<Rule style="active">
  <!-- for all ?x such that ?x is John -->
  <quantification> <!-- explicit quantification -->
    <Forall>
      <declare><Var>x</Var></declare><!-- for all ?x -->
      <guard><Equal><Var>x</Var><Ind>John</Ind></Equal></guard> <!-- such that ?x is John -->
    </Forall>
  </quantification>
  <if>...</if>
  <do>...</do>
</Rule>
```

Note that not only Rules but also facts, conjunctions, disjunctions, and many other constructs can be quantified in Reaction RuleML using the quantification element. In particular, quantified variable declarations with explicit variable bindings are also possible in the `<do>` part of a production rule in order to initialize local action variables for the actions.

2.2 ECA Reaction RuleML

In contrast to production rules, Event-Condition-Action (ECA) rules define an explicit event part which is separated from the conditions and actions of the rule. Their essential syntax is *on Event if Condition do Action*. ECA RuleML syntactically extends Production RuleML with an explicit `<on>` event part and rich (complex) event and action constructs and semantics defined in event/action libraries. Variants of this standard ECA rule are, e.g., Event-Action triggers (EA rules) and ECAP rules (ECA rules with postconditions after the action part).

```
<!-- ECA rule -->          <!-- EA trigger rule -->          <!-- ECAP rule with postcondition -->
<Rule style="active">    <Rule style="active">    <Rule style="active">
  <on>***</on>          <on>***</on>          <on>***</on>
  <if>...</if>          <on>***</on>          <if>...</if>
  <do>---</do>         <do>---</do>          <do>---</do>
</Rule>                </Rule>                <after>___</after>
                        </Rule>                </Rule>
```

We modify our discount example as follows:

```
<!-- On the order event of a product from a customer, if the customer is premium
do offer a discount of 5 percent to the customer -->
<Rule style="active">
  <on>
    <Event>
      <signature>
        <Atom><Rel per="value">order</Rel><Var>cust</Var><Var>prod</Var></Atom>
      </signature>
    </Event>
```

```

</on>
<if>
  <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
</if>
<do>
  <Action>
    <signature>
      <Atom><Rel per="effect">offer</Rel><Var>cust</Var><Var>prod</Var><Data>5.0</Data></Atom>
    </signature>
  </Action>
</do>
</Rule>

```

With the @per attribute it can be indicated whether the atomic <Rel>ations will be uninterpreted ("copy"), interpreted ("value"), effectful ("effect") or modal ("modal").

<Event> and <Action> (already introduced in Production RuleML) are used to represent events and actions. We distinguish between the <signature> of the event pattern definition, the concrete event instance <content> and other properties of the event (similar to the <Rule> properties).

```

<Event @key @keyref @iri @type>
  <!-- event info and life cycle management, modularization -->
  <oid> <!-- R: event instance object id --> </oid>
  <meta> <!-- R: (semantic) metadata of the event --> </meta>
  <scope> <!-- R: scope of the event --> </scope>
  <!-- event pattern description -->
  <evaluation> <!-- R: semantics: selection, consumption policies --> </evaluation>
  <signature> <!-- R: event pattern declaration --> </signature>
  <!-- event instance -->
  <qualification> <!-- R: e.g. qualifying event declarations, e.g.
    priorities, validity, strategy --> </qualification>
  <quantification> <!-- R: quantifying rule declarations --> </quantification>
  <content> <!-- R: event instance content --> </content>
</Event>

```

The following standard library defines a set of typical event, action, counting, temporal, and interval algebra operators for defining, e.g., complex events, actions, and intervals:

Action Algebra

Succession (Ordered), Choice (Non-Deterministic Choice),
Flow (Parallel Flow), Loop (Loops), Operator (generic Operator)

Event Algebra

Sequence (Ordered), Disjunction (Or), Xor (Mutually Exclusive), Conjunction (And),
Concurrent (Parallel), Any, Aperiodic, Periodic, Operator (generic Operator)

Counting Algebra

Counter, AtLeast, AtMost, Nth, Operator (generic Operator)

Negation / Absence Algebra

Not, Unless, Operator (generic Operator)

Temporal operators

Timer, Every, After, Any, Operator (generic Operator)

Interval Algebra (Time/Spatial/Event/Action Intervals)

During, Overlaps, Starts, Precedes, Succeeds, Meets,
Equals, Finishes, Operator (generic Operator)

With its typed logic, RuleML provides support for (re)using external temporal, spatial, situation, event, and action ontologies and metamodels which can be applied in the definition of semantic event/action types and temporal and spatial relations (see section [3.1](#)). Reaction RuleML defines generic elements such as `Event`, `Action`, `Situation`, `Time`, `Location`, `Interval`, `Operator`. The type of these generic elements can be defined by an `@type` reference to such external ontologies, e.g. to the Reaction RuleML metamodel. For instance, `<Operator type="ruleml:Sequence">` instead of `<Sequence>`. The following example shows a complex event pattern definition.

```
<Event key="ce2" type="ruleml:ComplexEvent">
  <signature>
    <Sequence>
      <!-- atomic event -->
      <signature>
        <Event type="ruleml:SimpleEvent">
          <signature><Atom>...event_A...</Atom></signature>
        </Event>
      </signature>
      <!-- nested complex event referenced by @keyref -->
      <signature><Event type="ruleml:ComplexEvent" keyref="ce1"/></signature>
      <!-- Common Base event selected via xpointer/xpath query in iri attribute -->
      <signature>
        <Event type="cbe:CommonBaseEvent" iri="cbe.xml#xpointer(//CommonBaseEvent)"/>
      </signature>
    </Sequence>
  </signature>
</Event>

<Event key="ce1">
  <signature>
    <Concurrent>
      <Event><meta><Time>...t3</Time></meta><signature>...event_B</signature></Event>
      <Event><meta><Time>...t3</Time></meta><signature>...event_C</signature></Event>
    </Concurrent>
  </signature>
</Event>
```

2.3 CEP RuleML

Complex Event Processing (CEP) is about the detection of complex events and reaction to complex events in near realtime [\[7\]](#). CEP rules might adopt the style of ECA rules in CEP RuleML, where the `<on>` event part is a complex event type definition; or, they might adopt the style of CA production rules where the (complex) event types are defined as restrictions on the variable binding definitions in the rule quantifications while event detection is then done in the condition part with the conditions representing the event pattern definitions. However, it is also possible to represent serial messaging CEP reaction rules which `<Receive>` and `<Send>` events in arbitrary combinations, leading to an event processing workflow-style logic following a forward-directed serial control flow for the literals in the rule. A serial (messaging) reaction rule starts either with a receiving event *on* – the trigger of the *global* reaction rule – or with a rule conclusion *then* – the queryable head literal of another messaging reaction rule or derivation rule (without further events or actions) – followed by an arbitrary combination of conditions *if*, events (*Receive*), and actions (*Send*) in the body

of the rule. This approach allows to combine forward-directed serial workflow-style execution of reaction rules with backward-reasoning derivation rules.⁷ This flexibility in terms of representing rule-based workflow style branching logics with a serial execution control flow for the rule literals and with support for modularization and aspect-oriented weaving of reactive rule code in combination with derivation rule code is in particular useful in distributed systems where event processing agents communicate and form a distributed event processing network, as e.g. in the following example:

```
<Rule style="messaging">
  <on><Receive> receive event from agent 1 </Receive></on>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
  <if> prove some conditions, e.g. make decisions on the received data
    using backward-reasoning derivation rules and/or </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
```

For better modularization the sub-conversation can be also written with a second reaction rule as follows:

```
<Rule>
  <on><Receive> receive event from agent 1 </Receive></on>
  <if> <!-- this goal activates the reaction rule via backward reasoning -- see below -->
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
<!-- the "if" goal from the first rule applies to the "then" conclusion of the second rule
and activates it via backward reasoning (rule chaining) -->
<Rule>
  <then>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </then>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
</Rule>
```

In the example the first reaction rule is triggered by a received event (**on**) which starts the forward-directed serial execution flow of the rule. It then deductively proves the condition (**if**) by applying it as a backward-reasoning subgoal on the knowledge base. This subgoal unifies with the head (**then**) of the second rule (which can be a standard derivation rule or a serial messaging reaction rule with further send and receive subgoal literals in the body). The second rule starts a subconversation sending a query to another agent 2 (**do**) and on receiving an answer from this agent (**on**) the execution of the second rule terminates successfully, which means the backward-reasoning subgoal of the first rule evaluates to true with the variable bindings for the variable **prod**. The first reaction rule then proceeds with its serial execution flow using the variable bindings for sending the results back to agent 1 (**do**) – either one by one in multiple messages for each resulting variable binding (default semantics according to backward-reasoning resolution semantics) or as a collection of all resulting answers (like in the `findall` built-in of Prolog or in join semantics of workflow branching logics).

⁷ See Prova's serial messaging rules; <http://www.prova.ws/>

That is, the semantics of this serial reaction rules combine standard logic programming (with resolution, backtracking, and variable binding) on the literals in combination with a serial forward directed execution of the rule literals which includes receiving and sending event queries to other distributed inference services and event processing agents (see section 3.3). This expressiveness allows us to represent workflow logics with parallel execution branches. The messaging reaction rules can be translated, e.g., into serial messaging Horn rules and executed in the Prova⁸ rule engine.

2.4 KR Reaction RuleML

Event/action logics, which have their origins in the area of knowledge representation (KR), focus on inferences that can be made from happened or planned events/actions, i.e. they describe inferences about the effects of events/actions on changeable properties of the world (situations, states). That is, the focus of this calculi typically is on the logical inferences and effects of events/actions without actually effecting any events/actions as in active reaction rules (although there can be combinations of the logical effects and the actual execution of actions and events). Reaction RuleML also defines syntax and semantics for knowledge representation event/action calculi such as Situation Calculus, Event Calculus, and Temporal Action Languages, etc. (for an overview see [14]) Specifically the notion of an explicit `<Situation>` (a.k.a. as state or fluent in Event Calculus) is introduced in KR Reaction RuleML. Situations are changeable fluents which are initiated or terminated as the effect of events. An event, which `<Happens>`, can `<Initiate>` or `<Terminate>` a situation. That is, a situation explicitly represents the abstract effect of occurred events and executed actions. Such states can e.g. be used for situation reasoning, e.g. in the condition part of reaction rules.

```

<!-- Initiates a situation --> <!-- Terminates a situation --> <!-- event happens -->
<Rule> <Rule> <Happens>*</Happens>
<on>*</on> <on>*</on>
<if>.</if> <if>.</if> <!-- situation holds? -->
<do><Initiate>-</Initiate></do> <do><Terminate>-</Terminate></do> <if><Holds>-</Holds></if>
</Rule> </Rule>

```

3 Reaction RuleML Features

Reaction RuleML provides several layers of expressiveness for adequately representing reactive logic and for interchanging events (queries, actions, event data) and rules. In the following, some of the expressive constructs of Reaction RuleML 1.0 are described.

3.1 Reaction RuleML Metamodel, Semantic Types and Data Queries

Reaction RuleML is based on metamodels and ‘pluggable’ ontologies. Figure 11 shows the top level structure of the Reaction RuleML metamodel. The Reaction

⁸ <http://prova.ws>

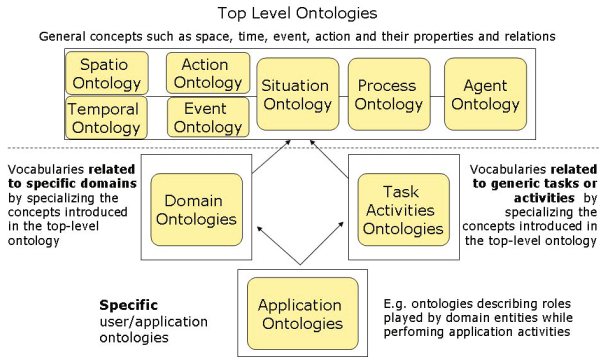


Fig. 1. Reaction RuleML Semantic CEP Metamodel

RuleML metamodel defines general concepts such as space, time, event, action situation, process, and agent in a modularized ontological top-level structure, with a left to right vertical order in the top-level ontologies. For instance, the concepts and relations for time and space are used in the event and action ontology, which is employed in the situation ontology etc. These general concepts defined in the top-level ontologies can be further specialized with existing domain ontologies and ontologies for generic tasks and activities (e.g. situation processing, processes/workflows, agents including their pragmatic protocols etc.). The applications ontologies for specialize these domain and task concepts with respect to a specific application, often on a more technical platform specific level. For instance, figure 2 shows the top-level ontology for situations. A situation description `hasProperties` defined by other ontologies such as time, event, etc., and `hasContent` with the main content of a situation description. A situation can be distinguished into two heterogeneous situation types – homogenous situation and heterogeneous situation. These two types can be further specialized in domain-specific ontologies with more specific situation types as illustrated in the figure.

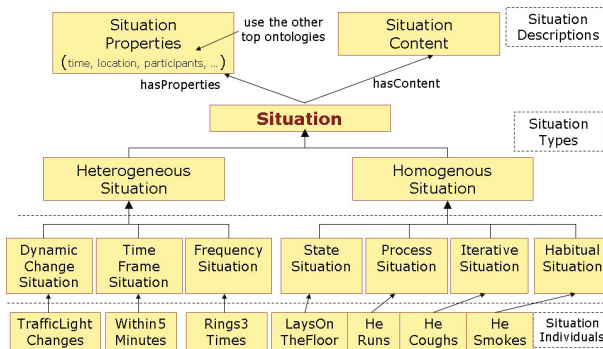


Fig. 2. Reaction RuleML Situation Metamodel

There are many existing ontologies which can be used in this generic modularized Reaction RuleML metamodel. For instance, the ontological constructs from the metamodel for the Bunge-Wand-Web representation model from [17] such as **Conceivable State Space**, **Lawful State Space**, **State Law**, **Stable State**, **Unstable State**, **History**, and further specializations of the top-level concepts such as event, thing, system, etc.

The Reaction RuleML metamodel defines the event, action, and interval algebra operators for complex events, actions, and intervals as well as other semantic concepts in the respective ontologies. These semantic definitions allow distinguishing, e.g., event/action algebra operators from interval operators and from logical operators (logical connectives). As described in section 2.2 Reaction RuleML comes with a library of predefined syntactic constructs which are syntactic short cut notations in the Reaction RuleML syntax for a more generic syntax element which points to its (semantic) type definition in the Reaction RuleML metamodel (or uses other external ontological definitions). To allow these external type definitions Reaction RuleML uses the `@type` attribute in the generic Reaction RuleML elements such as `Event`, `Action`, `Situation`, `Time`, `Location`, `Interval`, `Operator`, `Neg`, `Quantifier` etc., as illustrated by the following examples:

```
<Operator type="ruleml:Conjunction"> == <Conjunction>
<Negation type="ruleml:InflationaryNegation"> == <Naf> (in production rules)
<Action type="ruleml:Assert"> == <Assert>
<Event type="ruleml:SimpleEvent"> == <Atom> ... </Atom>
<Event type="ibm:CommonBaseEvent"> == IBM CBE
<Operator type="snoop:Sequence"> == Snoop == <Operator type="ruleml:Sequence"> == <Sequence>
```

While `@type` is used to refer to external type definitions (defined e.g. in external ontologies), the `@iri` attribute can be used as a general pointer to a (Web) resource having the semantics of containing either a standard IRI with a possible query specification (after the “?”) or an XPointer and XPath expression as manipulation and query language to point into and select data from external XML data sources. Note that XPointers and normal IRI references are syntactically distinguishable so that there is no problem in processing them differently.

```
<Atom>
  <Rel>name</Rel>
  <Ind iri="person.xml#xpointer(//Person/LastName[1]/text())"/>
</Atom>
```

It is possible to define XPointer-based query and manipulation expressions that operate on (large) resource sets instead on singleton resources, e.g. to specify a constructive view over a set of external nodes specified by an XPath query expression. The following example selects four `CommonBaseEvent` from an XML document.

```
<Event iri="cbe.xml#xpointer(//CommonBaseEvent[1]/range-to(//CommonBaseEvent[4]))"/>
<Action iri="BPEL.xml#xpointer(//invoke[@name=checkHotel])"/>
```

That is the IRIs are query expressions that return constructive views on resource sets that are treated as singletons, i.e. a constructive view over a set of

resources is a singleton with a unique IRI denoted by the query expression. It is possible to assign these views to variables and reuse the variables in the local reasoning/execution scope.

This external data and semantic knowledge can be used for Semantic Complex Event Processing (SCEP). In [19] we showed how the usage of semantic background knowledge about events and other related concepts can improve the quality of event processing. We described how to formalize semantic complex event patterns based on a logical interval-based event algebra, namely the interval-based Event Calculus [8, 9].

3.2 Rule Interface Descriptions with Semantic Profiles and Signatures

Reaction RuleML distinguishes between the interface of a rule and its implementation. The interface describes the functional and non-functional (semantic) properties of the rule. Reaction RuleML provides an interface definition language for the <signatures>s of publicly accessible rule functions together with their mode and type declarations. Modes are states of instantiation of the predicate/function described by mode declarations, i.e. declarations of the intended input-output constellations of the predicate terms with the following semantics:

- "+" The term is intended to be input
- "-" The term is intended to be output
- "?" The term is undefined/arbitrary (input or output)

For instance, the interface definition for the function $add(Result, Arg1, Arg2)$ is $add(-, +, +)$, i.e. the function predicate add returns one output argument followed by two input arguments. Also, all arguments must be integer values. Serialized in Reaction RuleML this would be:

```
<signature>
  <Atom>
    <Rel>add</Rel>
    <Var mode="-" type="xs:integer"/>
    <Var mode="+" type="xs:integer"/>
    <Var mode="+" type="xs:integer"/>
  </Atom>
</signature>
```

The following example illustrates the use of such signatures declarations in the interface descriptions of rules and distinguishes the interface from the implementation.

```
<!-- rule interface with two alternative interpretation semantics and a signature.
The interface references the implementation identified by the corresponding key -->
<Rule keyref="r1">
  <evaluation index="1">
    <!-- WFS semantic profile define in the metamodel -->
    <Profile type="ruleml:Well-Founded-Semantics" direction="backward"/>
  </evaluation>
  <evaluation index="2">
    <!-- alternative ASS semantic profile define in the metamodel -->
    <Profile type="ruleml:Answer-Set-Semantics" direction="backward"/>
  </evaluation>
</Rule>
```

```

</evaluation>
<!-- the signature defines the queryable head of the backward-reasoning rule -->
<signature>
  <Atom><Rel>discount</Rel><Var mode="-"/><Var mode="?"/><Var mode="+"/></Atom>
</signature>
</Rule>

<!-- implementation of rule 1 which is interpreted either by WFS or by ASS semantics
according to it's interface definition. -->
<Rule key="r1" style="reasoning">
  <if>
    <And>
      <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
      <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
    </And>
  </if>
  <then>
    <Atom><Rel>discount</Rel><Data>5.0</Data><Var>cust</Var><Var>prod</Var></Atom>
  </then>
</Rule>

```

The example above defines two rule interfaces which reference the implementation of rule 1 and rule 2 via **key-keyref** attributes. This approach supports modularization of the knowledge base and reusability of (XML) code by reference. By separating the interface description from the rule implementation, information hiding can be realized in distributed KBs. This is in particular useful for distributed rule inference service and rule agents which might publish some of the rule interfaces of their KBs publicly on the Web, but hide the concrete implementation of the rules. This enables a loosely-coupled interaction with the inference service / agent, where queries can be posed against the public interface signature **descriptions**. The interface defines the applicable evaluation semantics, which in the example uses predefined semantic **Profiles** from the RuleML metamodel.

Different interpretation, selection, consumption, and (transactional) execution policies for (complex) events and actions can be specified in the rule's **<evaluation>** semantics and the complex event/action descriptions.

The **<Profile>** can either point to externally defined profile type using **@type** (e.g. in the RuleML metamodel) or using **@iri** for general pointers to external definitions. Furthermore, since **<Profile>**'s content model is of type **xs:any** an arbitrary XML based definition of a profile can be specified here. This gives maximum flexibility in defining application-specific interpretation semantics and execution policies.

3.3 Reaction RuleML Messaging

As described in the previous section, the interface description language of Reaction RuleML allows for loosely-coupled interaction⁹ with distributed Reaction RuleML inference services and agent's KBs. For the communication between distributed rule-based (agent) systems, Reaction RuleML provides a general message syntax:

⁹ Decoupled interaction is also possible by just publishing event (messages), e.g. in an event stream or publish-scribe middleware.

```

<Message directive="PRAGMATIC CONTEXT" >
  <oid>          <!-- conversation ID-->          </oid>
  <protocol>     <!-- transport protocol -->      </protocol>
  <sender>       <!-- sender agent/service -->    </sender>
  <receiver>     <!-- receiver agent/service -->  </receiver>
  <content>      <!-- message payload -->        </content>
</Message>

```

In the context of these Reaction RuleML messages agents can interchange events (e.g., queries and answers) as well as complete rule bases (rule set modules), e.g. for remote parallel task processing. Agents can be engaged in long running possibly asynchronous conversations and nested sub-conversations using the conversation id to manage the conversation state. The protocol is used to define the message passing and coordination protocol. The directive attribute corresponds to the pragmatic instruction, e.g. a FIPA ACL primitive, i.e. the pragmatic characterization of the message context broadly characterizing the meaning of the message.

For sending and receiving (event) messages, Reaction RuleML 1.0 supports serial messaging CEP reaction rules that `<Receive>` and `<Send>` events in arbitrary combinations. A serial (messaging) reaction rule starts with a receiving event (`<on>`) followed by any combination of conditions (`<if>`), events (`<Receive>`), and actions (`<Send>`) in the body of the rule for expressing complex event processing logic. This flexibility with support for modularization and aspect-oriented weaving of reactive rule code is in particular useful in distributed systems where event processing agents communicate and form a distributed event processing network, as e.g. in the following example:

```

<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
  <if> prove some conditions, e.g. make decisions on the received data </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>

```

For better modularization the sub-conversation logic can be also written with an inlined reaction rule as follows:

```

<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <if> <!-- this goal activates the inlined reaction rule -- see below -->
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>

<Rule style="active">
  <then>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </then>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
</Rule>

```

4 Comparison with Other Reaction Rule Representations

In this section we discuss and analyze related representation languages for reaction rules. We focus on standards which are on the same modeling level as Reaction RuleML – that is, the Platform Independent Model (PIM) according to OMG’s Model Driven Architecture (MDA).

For an overview on standardizations see [16]. For a discussion of rule markup languages and reaction rule languages see [12, 11]. Current standardization efforts are also under way in the Event Processing Technical Society (EPTS) on a common event processing glossary and vocabulary as well as reference architectures and design patterns [15]. Besides many existing ontologies for events, time, space etc., and industry standards for event communication, there have been also many different approaches for rule-based event processing and reaction rule languages [14].

Based on the metamodel of the Bunge-Wand-Weber (BWW) representation model [17] we analyze the representational completeness of Reaction RuleML 1.0 with respect to an ontological reference metamodel and compare it with the following rule standards: OMG PRR, W3C SWRL, and W3C RIF. We also add Derivation RuleML 1.0 to this comparative coverage assessment, in order to show the differences and additional expressiveness of Reaction RuleML. Before we describe the analysis methodology and the analysis results we first introduce the languages.

W3C SWRL. The Semantic Web Rule Language (SWRL)¹⁰ is defined as a language combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (Unary/Binary Datalog). Rules in SWRL are of the form of an implication between an antecedent (body) conjunction and a consequent (head) conjunction, where description logic expressions can occur on both sides. The intended interpretation is as in classical first-order logic: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Relationships between RuleML and SWRL: The W3C member submission SWRL combines an earlier version of RuleML with OWL 1.0. An effort was recently started to update SWRL, including for RuleML 1.0 and OWL 2.0.

W3C RIF. The W3C Rule Interchange Format (RIF) Working Group¹¹ is an effort, influenced by RuleML, to define a standard RIF for facilitating the exchange of rule sets among different systems and to facilitate the development of intelligent rule-based application for the Semantic Web. So far, the RIF Working Group has published two dialects as recommendations – the Production Rules Dialect (PRD) for non-ground production rules with inflationary negation and the Basic Logic Dialect (RIF-BLD), which semantically corresponds to a Horn rule language with equality but without negation. A common condition language

¹⁰ <http://www.w3.org/Submission/SWRL/>

¹¹ http://www.w3.org/2005/rules/wiki/RIF_Working_Group

(RIF-Core) is shared between these two dialects. Like RuleML, RIF-BLD has a number of syntactic extensions with respect to 'regular' Horn rules, including F-logic-like frames, and a standard system of built-ins drawn from Datatypes and Built-Ins (RIF-DTB). The connection to other W3C Semantic Web languages is established via RDF and OWL Compatibility (RIF-SWC).

Relationships between Production RuleML and RIF-PRD: Members of the Reaction RuleML Technical Group have co-edited the W3C RIF Production Rule Dialect (RIF-PRD). RIF-PRD with inflationary negation is a less expressive subset of PR RuleML. Syntactically, production rules in RIF-PRD are written in `if-then` syntax instead of PR RuleML's `if-do` syntax, the latter allowing a clear semantic distinction of a conclusion (`then` part) and an action (`do` part), e.g. when both are allowed for the same rule. In RIF-PRD `Do` is used as a type tag to syntactically denote a compound action which is a sequence of standard production rule actions (`Assert`, `Retract`, and `Modify`), whereas Reaction RuleML supports expressive complex action definitions using action algebra operators. Quantifying variable binding declarations are supported by RIF-PRD (`declare`) and by Production RuleML (quantification), which in addition also supports rule qualifications.

OMG PRR. OMG's Production Rule Representation (PRR)^[12] is a modeling language for production rules. It uses OMG MOF to define a generic meta-model for production rules and extends UML for modeling production rules. PRR includes two types of rules: Forward chaining inference rules (e.g. Rete) and sequentially processed procedural rules. PRR is defined at two levels. The adopted PRR Core of PRR 1.0 includes the general rule and production rule model. The PRR OCL, which is currently also in the focus of the PRR 1.1 effort, includes an extended OCL expression language enabling compatibility with non UML representations.

Relationships between Production RuleML and OMG PRR: Based on [20] and Production RuleML, members of the Reaction RuleML Technical Group have co-edited the OMG Production Rule Representation (PRR). RuleML is one of the languages whose features are to be covered by PRR on an abstract level. Since PRR is a meta-language, Production RuleML's XML syntax can be used as a concrete expression language instantiating PRR models. That is, OMG PRR provides a way to include rules into the (UML) model of an application at design time and Production RuleML then provides a standard means of translating the model and feeding the executable rules into a PR application at run time.

Event Processing Standards. Besides Reaction RuleML, there exist no overarching interchange and representation standard for reaction rules and rule-based event processing. Although several of the standards efforts such as PRR and RIF (a Reaction Rules Dialect has been started but not finalized in RIF 1.0) have reaction rules on their long-term agenda, no concrete standardization effort started yet [16]. There have been several proposals for ECA-style rule languages

¹² <http://www.omg.org/spec/PRR/1.0/PDF/>

and platform-specific languages coming from research projects (Rewerse R2ML, ...) or from industry communities (IBM SRML, CBE, ...), partially defined as XML languages. Several event ontologies have been proposed.

Relationships between CEP RuleML and EPTS work: RuleML is a founding member of the Event Processing Technical Society (EPTS)¹³. Members of the Reaction RuleML Technical Group are contributing to the work on an Event Processing glossary, event processing language models, use cases, reference architectures, and design patterns.^[15] With its flexible and extensible approach, CEP RuleML is a highly expressive rule-based Event Processing Language (rule-based EPL) which can make use of external event and action metamodels / ontologies such as the many existing event ontologies or the planned OMG Event Metamodel and Profile (EMP). Since CEP RuleML syntactically builds on top of Production RuleML and ECA RuleML – besides flexible (messaging) reaction rules – both major rule types can be used for representing (complex) event processing rules. Moreover, CEP RuleML can adequately represent typical use cases and functionalities in Event-Driven Architectures (EDAs) and (distributed) EP architectures.

For the analysis we selected a set of relevant ontological constructs from the BWW representation model (BWW RM) and classified them according to typical top-level categories of expressiveness for reaction rules. We slightly adapt the BWW model, introducing e.g. an explicit **Action** concept instead of an **Event** which **acts on**. We also introduce the concept of an **Agent** instead of the un-specific concept **System**. Following the methodology of [18] we then used this as a reference model for our analysis by comparing the representation constructs of the analyzed rule languages to the representation constructs of the reference model. We analyze these results according to the (ontological/metamodel) completeness and clarity of the rule representation / modeling languages. Therefore, we identify the representation deficiencies using the following metrics: *construct deficit* (with respect to a construct in the BWW reference model), *redundancy* (multiple language constructs map to the same construct in the BWW reference model), *overload* (the same language construct maps to multiple (semantically different) constructs in the BWW reference mode), and *excess* (language constructs that do not have a mapping to the BWW reference model). Table 1 shows the results of the comparison with the top-level categories. At this top level, Reaction RuleML shows high ontological completeness and clarity with respect to the adapted BWW reference model, which is due to its close semantic

Table 1. Comparison of Rule Markup and Modeling Standards

<i>BWWRM</i>	<i>SWRL</i>	<i>PRR</i>	<i>RIF</i>	<i>DerivationRuleML</i>	<i>ReactionRuleML</i>
Thing	+	-	+	+	+
Event	-	-	-	-	+
Action	-	+	+	-	+
State (Situation)	-	-	-	-	+
System (Agent)	-	-	-	-	+

¹³ <http://www.ep-ts.com/>

similarity to the top level constructs in the Reaction RuleML metamodel. The **System** perspective in Reaction RuleML relates to the **Agent** metamodel in Reaction RuleML, where Reaction RuleML serves as a standardized rule and event interchange format between the agents / inference services (including the agent's interface descriptions). The agents internally run platform-specific rule engines such as Prova, OO jDREW, Drools, and Emerald¹⁴ – see e.g. the Rule Responder project¹³¹⁵ for a reference implementation. Besides its use for smart reactive agents on the Web, future application domains of Reaction RuleML might directly enrich Rich Internet Applications (RIA), which are then translated into a domain-specific (Web browser) reaction rule language such as JSON Rules⁴¹⁶ Translators, such as the Reaction RuleML translator service framework and the MYNG syntax configurator¹⁷ of RuleML's Relax NG schemas¹¹, contribute to the interoperability by translating from the platform specific languages into (Reaction) RuleML and vice versa, as well as between RuleML/XML and other XML-based languages such as RIF/XML. RIF RuleML interoperation was started with a common subset². Other systems include rule editors (e.g., Acumen Business Rule Manager and S2REd¹⁸).

5 Conclusion

The four major reactive rule types were surveyed based on Reaction RuleML. Its uniform and flexible syntax can be specialized to branches of the corresponding subfamily of RuleML. With its metamodel and ability to plug in external vocabularies and type systems, Reaction RuleML has been developed as an extensible semantic rule language that is highly configurable via external semantic definitions. Reaction RuleML supports distributed and modularized knowledge bases through direct coupling via key references within a KB, `iri` pointers, and support for query languages. With messaging, it also supports loosely-coupled interface-based interaction using rule signatures and decoupled communication via event messages. The specification of Reaction RuleML 1.0 is being developed bi-schematically, 'Rosetta'-style, in XML Schema and Relax NG.

References

1. Athan, T., Boley, H.: Design and implementation of highly modular schemas for XML: Customization of ruleML in relax NG. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 17–32. Springer, Heidelberg (2011)
2. Boley, H.: RIF RuleML Rosetta Ring: Round-Tripping the Dlex Subset of Datalog RuleML and RIF-Core. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 29–42. Springer, Heidelberg (2009)

¹⁴ <http://www.prova.ws>; <http://www.jdrew.org/ojdrew>;
<http://www.jboss.org/drools/>; <http://lpis.csd.auth.gr/systems/emerald/>

¹⁵ <http://responder.ruleml.org>

¹⁶ See e.g. RuleTheWeb Firefox extension <http://ruletheweb.org> ⁵

¹⁷ <http://ruleml.org/1.0/myng/>

¹⁸ <http://www.acumenbusiness.com> and <http://sourceforge.net/projects/s2red>

3. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: The Overarching Specification of Web Rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010)
4. Giurca, A., Pascalau, E.: JSON Rules. In: CEUR Workshop Proceedings of 4th Knowledge Engineering and Software Engineering (KESE), vol. 425 (2008)
5. Giurca, A., Tylkowski, M., Müller, M.: RuleTheWeb: Rule-Based Adaptive User Experience. In: CEUR Workshop Proceedings of the 6th International Rule Challenge at RuleML 2012 (to appear, 2012)
6. Kowalski, R., Sadri, F.: A Logic-Based Framework for Reactive Systems. In: Bikakis, A., Giurca, A. (eds.) RuleML 2012. LNCS, vol. 7438, pp. 1–15. Springer, Heidelberg (2012)
7. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman, Amsterdam (2002)
8. Paschke, A.: ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language. In: RuleML 2006, Athens, Georgia, USA (2006)
9. Paschke, A.: ECA-RuleML: An Approach Combining ECA Rules with Temporal Interval-Based KR Event/Action Logics and Transactional Update Logics. CoRR, abs/cs/0610167 (2006)
10. Paschke, A.: Rules and Logic Programming for the Web. In: Polleres, A., d’Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 326–381. Springer, Heidelberg (2011)
11. Paschke, A., Boley, H.: Rule Markup Languages and Semantic Web Rule Languages. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 1–24. IGI Publishing (May 2009)
12. Paschke, A., Boley, H.: Rules Capturing Events and Reactivity. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies, pp. 215–252. IGI Publishing (May 2009)
13. Paschke, A., Boley, H.: Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web. *Int’l Journal Artificial Intelligence Tools* 20(6), 1043–1081 (2011)
14. Paschke, A., Kozlenkov, A.: Rule-Based Event Processing and Reaction Rules. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 53–66. Springer, Heidelberg (2009)
15. Paschke, A., Vincent, P., Alves, A., C., Moxey: Tutorial on Advanced Design Patterns in Event Processing. In: DEBS (2012)
16. Paschke, A., Vincent, P., Springer, F.: Standards for complex event processing and reaction rules. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 128–139. Springer, Heidelberg (2011)
17. Rosemann, M., Green, P.: Developing a Meta Model for the Bunge-Wand-Weber Ontological Constructs. *Inf. Syst.* 27(2), 75–91 (2002)
18. Rosemann, M., Green, P., Indulska, M.: A Reference Methodology for Conducting Ontological Analyses. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 110–121. Springer, Heidelberg (2004)
19. Teymourian, K., Paschke, A.: Semantic Rule-Based Complex Event Processing. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 82–92. Springer, Heidelberg (2009)
20. Wagner, G., Antoniou, G., Tabet, S., Boley, H.: The Abstract Syntax of RuleML – Towards a General Web Rule Language Framework. In: Web Intelligence, pp. 628–631. IEEE Computer Society (2004)

A Production Rule-Based Framework for Causal and Epistemic Reasoning^{*}

Theodore Patkos¹, Abdelghani Chibani¹,
Dimitris Plexousakis², and Yacine Amirat¹

¹ Lissi Laboratory, University of Paris-Est Creteil
{patkos,chibani,amirat}@u-pec.fr

² Institute of Computer Science, FO.R.T.H.
dp@ics.forth.gr

Abstract. Action theories are an important field of knowledge representation for reasoning about change and causality in dynamic domains. In practical implementations agents often have incomplete knowledge about the environment and need to acquire information at runtime through sensing, the basic ontology of action theories needs to be extended with epistemic notions. This paper presents a production system that can perform online causal, temporal and epistemic reasoning based on the Event Calculus and on an epistemic extension of the latter. The framework implements the declarative semantics of the underlying logic theories in a forward-chaining rule-based system. This way, it combines the capacity of highly expressive formalisms to represent a multitude of commonsense phenomena with the efficiency of rule-based reasoning systems, which typically lack real semantics and high-level structures.

1 Introduction

Action theories have emerged as an important subfield of knowledge representation in Artificial Intelligence for reasoning about actions and causality in dynamic environments. Developed originally within the context of cognitive robotics, related formalisms have studied the relationship between the *knowledge*, the *perception* and the *action* of autonomous agents [1], in order to infuse them with cognitive and commonsense skills. A fundamental extension of most action theories, vital for real-world domains, is related with their ability to refer not only to what an agent knows, but also to what it does not know ([2], chapter 23). This requires the modeling of epistemic notions and an account of knowledge change through knowledge-producing (sense) actions. Epistemic extensions of action theories have been used to reason about a multitude of commonsense phenomena in partially observable domains, where agents deliberate about the effects of actions having incomplete knowledge about the preconditions and the state of the environment (e.g., [3–5]). To fully exploit the potential of epistemic

^{*} This work is partially supported by the ITEA2 09031 A2Nets and 10035 PREDYKOT projects.

reasoning in practice, a number of challenges need to be faced, with most important the ability to acquire information at run-time and effectively perform reasoning tasks on-the-fly. This requires the coupling of formal theories that are feasible under real-world conditions with efficient reasoning techniques.

In this paper, we present the design and implementation of a forward-chaining production system that can perform causal, temporal and epistemic reasoning both offline and online. The system combines rich declarative languages, such as the Event Calculus and the recently proposed epistemic extension Discrete-time Event Calculus Knowledge Theory (DECKT) [6], with features that enhance efficient online reasoning, such as rule-based operational semantics, semi-destructive knowledge update, negation-as-failure (NaF) and others. Whereas most implementations of the Event Calculus so far are based on logic programming, deductive or satisfiability techniques, which are mainly goal-driven, the proposed system follows the forward-chaining production rules paradigm, which has demonstrated prominent results for exhibiting reactive behavior at runtime.

The contribution of our approach is primarily pragmatic: in addition to an implementation of the Event Calculus suitable for reacting to occurring events, which is not supported by most current Event Calculus reasoners (with notable exception the system presented in [7]), we further achieve to transfer the benefits of the formalisms, such as the solution to the frame problem, temporal and epistemic reasoning, or multi-model generation due to non-determinism, into an efficient forward-chaining system. The proposed approach goes beyond ordinary rule-based systems deployed in dynamic domains, where the actions that lead to the assertion and retraction of facts have no real semantics and high-level structures. Instead, it uses the underlying structures of the Event Calculus and the Kripke equivalent semantics of DECKT to define the causal properties of actions or to manipulate ordinary and epistemic context-dependent facts. DECKT employs a sound and complete yet computationally less intensive representation for knowledge with respect to theories based on possible worlds structures.

The paper is structured as follows. After an introduction to the underlying formal languages, we elaborate in Section 3 on the features of the rule-based reasoner. Section 4 discusses implementation issues and Section 5 reports on the application of the reasoner in an open-world use-case. We conclude in Section 6.

2 Background

The reasoner implements the Event Calculus and the Discrete time Event Calculus Knowledge Theory (DECKT), an epistemic extension of the basic formalism.

2.1 The Discrete Time Event Calculus

The Event Calculus [8] is a narrative-based many-sorted first-order language for reasoning about action and change, where *events* indicate changes in the

environment, *fluents* denote time-varying properties and a *timepoint* sort implements a linear time structure. The calculus applies the *principle of inertia*, which captures the property that things tend to persist over time unless affected by some event. It also uses *circumscription* to solve the frame problem and support default reasoning. A set of predicates is defined to express which fluents hold when (*HoldsAt*), which events happen (*Happens*), which their effects are (*Initiates*, *Terminates*, *Releases*) and whether a fluent is subject to the law of inertia or released from it (*ReleasedAt*).

Our account of action and knowledge is formulated within the circumscriptive linear Discrete Event Calculus extensively described in [9]. The commonsense notions of persistence and causality are captured in a set of domain independent axioms, which we refer to as \mathcal{DEC} , that express the influence of events on fluents and the enforcement of inertia for the *HoldsAt* and *ReleasedAt* predicates.

A particular domain description consists of axioms that describe the commonsense domain of interest, observations of world properties at various times and a narrative of known world events [1].

Definition 1. (Event Calculus Domain Description) *A non-epistemic Event Calculus domain description $\mathcal{D} = \langle \Sigma, \Delta_2, \Psi, \Gamma, \Delta_1, \Omega \rangle$ consists of: - a set Σ of positive, negative and release effect axioms that describe the conditions under which an event e initiates, terminates or releases a fluent f at timepoint t , respectively:*

$$\begin{aligned} \bigwedge^{f_i \in C} [HoldsAt(f_i, t)] &\Rightarrow Initiates(e, f, t) \\ \bigwedge^{f_i \in C} [HoldsAt(f_i, t)] &\Rightarrow Terminates(e, f, t) \\ \bigwedge^{f_i \in C} [HoldsAt(f_i, t)] &\Rightarrow Releases(e, f, t) \end{aligned}$$

where C denotes the context of the axiom (the set of precondition fluents), i.e. $C = \{f_1, \dots, f_n\}$, $n \geq 0$.

- a set Ψ of state constraints: $\bigwedge^{f_i \in C_f} [HoldsAt(f_i, t)] \Rightarrow HoldsAt(f, t)$
 - a set Δ_2 of trigger axioms:

$$\bigwedge^{f_i \in C_e} [HoldsAt(f_i, t)] \wedge \bigwedge^{e_i} [Happens(e_i, t)] \Rightarrow Happens(e, t),$$

where C_e is the set of precondition fluents for the triggering of e
 - a set $\Gamma = \Gamma(0) \cup \dots$, denoting the observations at each timepoint,
 - a set $\Delta_1 = \Delta_1(0) \cup \dots$, denoting the narrative of actions, and
 - a set Ω of unique names axioms. □

Explanation closure axioms are created by means of circumscription to minimize the extension of all *Initiates*, *Terminates*, *Releases*, *Happens* predicates.

A *knowledge base* $KB(t)$ is a set of ground facts (i.e., fluents and events) and represents the state of the world at timepoint t and the events that are planned to occur at this timepoint. Initially, $KB(0) = \emptyset$.

¹ In the sequel, variables of the sort event are represented by e , fluent variables by f and variables of the timepoint sort by t , with subscripts where necessary. Free variables are implicitly universally quantified.

2.2 Epistemic Reasoning with the Event Calculus

DECKT [6, 10] is a provably sound and complete extension of the Event Calculus that introduces epistemic features enabling reasoning under partial observability about a wide range of commonsense phenomena, such as temporal and delayed knowledge effects, knowledge ramifications, non-determinism and others. It assumes agents acting in dynamic environments possessing accurate but potentially incomplete knowledge and able to perform sensing and actions with context-dependent effects. It introduces four epistemic fluents, namely *Knows*, *Kw* (for "knows whether"), *KP* (for "knows persistently") and *KPw*. Whenever knowledge is subject to inertia the *KP* fluent is used. The latter is related with the *Knows* fluent based on the axiom:

$$\text{(KT2)} \quad \text{HoldsAt}(KP(\phi), t) \Rightarrow \text{HoldsAt}(\text{Knows}(\phi), t),$$

where ϕ is a non-epistemic fluent formula. The *Knows* fluent expresses knowledge about domain fluents and formulae. Fluent formulae inside epistemic fluents are reified so that $\text{Knows}(P(x) \vee Q(x))$, for instance, is formally treated as a term of first-order logic, rather than an atom. We define the abbreviation $\text{HoldsAt}(Kw(\phi), t) \equiv \text{HoldsAt}(\text{Knows}(\phi), t) \vee \text{HoldsAt}(\text{Knows}(\neg\phi), t)$ (similarly for $\text{HoldsAt}(KPw(\phi), t)$)².

Instead of manipulating a set of possible worlds, the objective of DECKT is to extend a given domain axiomatization using meta-axioms that transform each effect axiom into a set of new ones, thus enabling an agent to perform epistemic derivations under incomplete information. For instance, for each *positive effect axiom* $\bigwedge^{f_i \in C} [\text{HoldsAt}(f_i, t)] \Rightarrow \text{Initiates}(e, f, t)$ DECKT introduces a statement expressing that if the conjunction of preconditions is known then after e the effect will be known:

$$\text{(KT3.1)} \quad \bigwedge^{f_i \in C} [\text{HoldsAt}(\text{Knows}(f_i), t)] \Rightarrow \text{Initiates}(e, KP(f), t)$$

However, if some precondition is unknown while none is known false, then after e knowledge about the effect is lost:

$$\text{(KT5.1)} \quad \neg \text{HoldsAt}(\text{Knows}(f), t) \wedge \bigvee^{f_i \in C} [\neg \text{HoldsAt}(Kw(f_i), t)] \wedge \\ \neg \text{HoldsAt}(\text{Knows}(\bigvee^{f_i \in C} \neg f_i), t) \Rightarrow \text{Terminates}(e, KPw(f), t)$$

In this case, though, an epistemic relation is also created among the unknown preconditions and the effect; if at some future timepoint the agent obtains information through sensing about the state of the preconditions, it may also infer the state of the effect:

$$\text{(KT6.1.1)} \quad \neg \text{HoldsAt}(\text{Knows}(\bigvee^{f_i \in C} \neg f_i), t) \wedge \bigvee^{f_i \in C} [\neg \text{HoldsAt}(Kw(f_i), t)] \wedge \\ \neg \text{HoldsAt}(\text{Knows}(f), t) \Rightarrow \text{Initiates}(e, KP(f \vee \bigvee^{f_j \in C(t)^-} \neg f_j), t),$$

² The abbreviation only refers to the *Kw* and *KPw* fluents inside the distinguished predicate *HoldsAt*; these epistemic fluents can still be used as ordinary fluents inside any other predicate of the calculus, e.g., $\text{Terminates}(e, KPw(f), t)$.

where $C(t)^- = \{f \in C \mid \neg \text{HoldsAt}(Kw(f), t)\}$ the set of precondition fluents that are unknown to the agent³. Informally, the theory is augmented with a disjunctive knowledge formula, called a *hidden causal dependency* (HCD) [6], that, considering (KT2), is equivalent to $\text{HoldsAt}(\text{Knows}(\bigwedge_{f_j \in C(t)^-} f_j \Rightarrow f), t + 1)$. In addition, when the effect is known to be false upon event occurrence, a HCD is created between the effect and each of the unknown to the agent fluents.

The intuition is analogous for *negative* and *release effect axioms*. HCDs denote temporal implication relations and are treated as ordinary fluent terms. DECKT defines, in addition to the creation of HCDs, also axioms that determine when the latter are destroyed and what knowledge should be preserved when a HCD is destroyed. An in depth description of the axiomatization is beyond the scope of this study (see [10]). We note only that HCDs characterize how an agent believes the state of the world has evolved at a particular time instant, while state constraints prescribe how the world should be at all times.

Finally, sense actions provide information about the truth value of fluents and, by definition, only affect the agent's mental state:

(KT4) *Initiates*(*sense*(f), $KPw(f)$, t)

3 A Rule-Based Production System for the Event Calculus

In this paper, we describe a rule-based reasoner for the Event Calculus that can perform causal, temporal and epistemic reasoning tasks utilizing information obtained at run-time. In our logic-based forward-chaining framework Event Calculus epistemic and non-epistemic axioms, state constraints and domain rules are compiled into production rules, preserving the declarative and Kripke equivalent semantics. In this section, we describe the operational semantics of the system's reasoning cycle that achieves run-time monitoring of a dynamic environment.

The knowledge base is structured as a deductive database. A rule engine matches facts in the working memory, event narratives and observations arriving on-the-fly with conditions of rules, deriving the resulting world state and the events that are or may be triggered. In contrast with ordinary rule-based systems deployed for reactive reasoning in dynamic worlds, where the actions that lead to the assertion and retraction of facts have no real semantics and high-level structures, our system is based on rich declarative languages. It uses the underlying high-level structures of the Event Calculus to define causal properties, as well as DECKT to distinguish between the ordinary and epistemic facts that are initiated, terminated or triggered based on the given context.

³ Note that although the set C is fixed for a particular domain, $C(t)^-$ is a dynamic set that changes from timepoint to timepoint. Axioms such as (KT6.1.1) are a compact way of representing inferences without having to enumerate all axioms where knowledge about fluents in C appear as preconditions. They are not meant to be parsed at design-time, but can be straightforwardly implemented using lists constructs in Prolog- or Lisp-like languages as explained in Section 4.

The operational semantics implement a model generator to construct all possible models that satisfy a given narrative, a set of causal and temporal constraints and observations obtained at runtime. This is particularly interesting when simulating the behavior of a system in the non-epistemic case, where non-determinism and released fluents may give rise to distinct models at each time instant. For the epistemic case on the other hand, DECKT meta-axioms capture knowledge change in a set of HCD implication rules, thus requiring only a single model to be preserved at all times. This leads to an efficient representation of knowledge change [10] that is well suited for rule-based implementations.

3.1 The Operational Semantics

Next, we define the reasoning cycle, in which the reasoning module repeatedly monitors the state of fluents, the execution of actions and the occurrence of events in the environment and updates the pool of KBs to reflect the changes brought about by these observations.

The assimilation of observations regarding the state of fluents requires careful treatment within an online system. The Event Calculus does not allow for inertial (i.e., not released) fluents to modify their state unless an event explicitly interacts with them. Consequently, for the non-epistemic component, where complete world description must be preserved at all times, observations about fluents that contradict stored derivations lead to model elimination. Updating knowledge is only applicable within the epistemic component where sense action are specifically used for that purpose.

The following reasoning cycle is executed both by omniscient systems that perform non-epistemic reasoning in a non-deterministic environment and by devices that have only limited access to sensor data and maintain a KB that describes only partially the current world state. A time threshold T determines how often information from the environment should be interleaved with execution.

Definition 2. (Reasoning cycle) *Given the domain-independent \mathcal{DEC} and DECKT axiomatizations, a domain description $\mathcal{D} = \langle \Sigma, \Delta_2, \Psi, \Gamma, \Delta_1, \Omega \rangle$ and a bound T on the number of reasoning steps (i.e., timepoints), the following reasoning cycle determines a sequence of state transitions $\langle PKB(t_0) \rangle, (\Gamma(t_0), \Delta_1(t_0)), \dots, \langle PKB(t_0 + T - 1) \rangle$, where $PKB(t')$ is the pool of knowledge bases $KB_j(t')$ that represent the valid models produced at timepoint t' and t_0 is the initial timepoint:*

- Step A (Init) *For all $t_0 \leq t < T$ take the next element $KB_j(t)$ from the pool of knowledge bases until $PKB(t)$ is empty.*
- Step B (Main Loop)
 - Step B.1 (Prog) *Apply \mathcal{DEC} and optionally DECKT axioms to $KB_j(t) \cup \Delta_1(t) \cup \Sigma$ to obtain all inertial fluents at $t + 1$. Let $KB_j^*(t'')$ denote the inertial fluents of $KB_j(t'')$ at some timepoint t'' , i.e., $KB_j^*(t'') = (KB_j(t'') - \{f : Released(f, t'')\})$. Then, obtain $KB_j^*(t + 1)$ as follows: $KB_j^*(t + 1) = (KB_j^*(t) - \{f : Happens(e, t) \in \Delta_1(t), Terminates(e, f, t) \vee Releases(e, f, t) \in \Sigma \text{ and all } f_i \in C \text{ hold in } KB_j(t)\}) \cup (\{f : Happens(e, t) \in \Delta_1(t), Initiates(e, f, t) \in \Sigma \text{ and all } f_i \in C \text{ hold in } KB_j(t)\})$.*

- Step B.2 (StC) *Apply state constraints of Ψ to obtain all indirect effects of actions as follows: $KB_j^{**}(t+1) = KB_j^*(t+1) \cup (\{f : \text{HoldsAt}(f, t+1) \text{ is in the head of some } r \in \Psi \text{ and all } f_i \in C_f \text{ of } r \text{ hold in } KB_j^*(t+1)\})$.*
- Step B.3 (NaF) *All fluents not stored in $KB_j^{**}(t+1)$ are assumed not to hold, thus forming $KB_j(t+1)$.*
- Step B.4 (Tr) *Apply trigger axioms in $\Delta_2(t+1)$ to obtain all occurring events as follows: $\Delta_1'(t+1) = \Delta_1(t+1) \cup (\{e : \text{Happens}(e, t+1) \text{ is in the head of some } r \in \Delta_2(t+1), \text{ all } f_i \in C_e \text{ of } r \text{ hold in } KB_j(t+1) \text{ and } \text{Happens}(e_i, t+1) \in \Delta_1(t+1)\})$. That is, for every instance r of a trigger axiom in $\Delta_2(t)$ such that its context and event occurrence preconditions are satisfied in $KB_j(t+1) \cup \Delta_1(t+1)$, assert the head of r in $\Delta_1'(t+1)$.*
- Step C (Expand) *Determine non-epistemic fluents that are released in the interval $[t, T)$ and are not in the head of an activated state constraint of Ψ . For every $t' \in [t, T)$ create all possible combinations of the truth values of these fluents and assert them in duplicates of $KB_j^{**}(t')$. If no state constraint is violated perform steps B.2-4 to obtain a new KB (new model) and store it in $PKB(t')$. Then, return to Step A.*
- Step D (Sense) *Update Γ and Δ_1 sets according to information obtained by external procedures (e.g., sensors, communication modules, actuators etc). For each element $KB_j(t)$ of the pool of knowledge bases, if $KB_j(t) \cup \Gamma(t)$ is inconsistent (observations contradict inferences) remove $KB_j(t)$ from $PKB(t)$.*

Notice that only positive fluents are asserted in the knowledge base at steps B.1 and B.2. Negative fluents are assumed to be false by application of NaF, therefore they need not be explicitly introduced. Yet, the modeling of state constraints and non-determinism deems necessary a semi-destructive update of the knowledge base; destructive assignment is applied only on inertial fluents at step B.1, preserving also the solution to the frame problem without the use of additional axioms. Indirect derivations and released fluents are asserted in successive steps of the execution.

3.2 Complexity Analysis

Theoretical computational complexity issues of the Event Calculus have been studied in the past with respect to a multitude of reasoning tasks (e.g., [11, 12]). Our implementation supports the subsets of the Event Calculus and DECKT introduced in Section 2, in order to perform temporal projection with events arriving at chronological order. In this section we report on the computational complexity of the reasoning cycle.

Steps B.1, B.4 evaluate precondition fluents to determine which effect or trigger axioms are activated. The problem of query answering on (untyped) ground facts (data complexity) reduces to the problem of unifying the query with the facts, which is $O(n)$, where n is the size of the KB. In the worst case, the number of fluents that need to be checked is n both in the epistemic and the non-epistemic case: only atomic fluent preconditions need to be considered.

Step B.2 carries out deductive closure (materialization) on ground facts on a set of universally quantified rules to produce all possible inferences that make

implicit information explicit. In the general case, different strategies can be deployed to accomplish this task characterized by diverse complexity properties. Within our framework logical inference at each step is performed by Jess⁴, an efficient rule engine that uses an enhanced version of the Rete algorithm to process rules. Jess makes intensive use of hash tables and caching to reduce time complexity of pattern matching, which is tightly dependent on the syntactic form of rules written by the knowledge engineer. In the best case, the performance of the reasoner is $O(p)$, i.e. linear to the number of patterns in a rule and independent of the size of the KB, whereas in the worst case of badly constructed rules and naive pattern-matching, it can be $O(p^n)$, where n the number of facts in the KB that the reasoner has to evaluate in order to determine which rules to trigger. Note also that in our case reasoning at each step operates in a monotonic space; no information that has been inferred for a given timepoint is deleted (facts retracted at Step B.1 refer to the successor timepoint). This enhances the behavior of the system, considering for instance that pattern-matching in Jess is performed while changes are made in the working memory, rather than just at the actual execution time.

As a result the size of the domain may play a decisive role for computing Step B.2. For a domain of n fluents the number of facts that influence performance are $O(n)$ at worse for the non-epistemic case and $O(2^n)$ for the epistemic case, due to the fact that DECKT treats all disjunctions as ordinary fluents. The number of state constraints plays no role in the complexity as Ψ is fixed and known at compilation time.

Step C is linear to the number of fluents for the non-epistemic case; a query needs to be issued to the KB and for each released fluent retrieved an implementation-specific cloning procedure is executed. For the epistemic case Step C is not applicable, as there is always a single KB to manage. This is also the case at Step A: for the non-epistemic deliberation there might be at worse an exponential number of KBs to the size of the domain at each timepoint, whereas DECKT axiomatizes an epistemic domain on a single KB. Finally, for each KB a query of linear cost needs to be issued at Step D as well, in order to compare the stored truth value of the sensed fluent with the input from sensing.

Conclusion: It becomes clear that the predominant complexity factor for the non-epistemic case is the number of released fluents causing the creation of KBs, whereas for the epistemic case it is the number of HCDs stored. The former characterizes the degree of non-determinism of the effects of actions in a given environment. The latter parameter, i.e., the set of HCDs that are created as an agent performs actions with unknown preconditions, is based on the knowledge that the agent has about the state of the world, as well as on the sets of state constraints that capture interrelated fluents. So called *dominos domains* which lead to chaotic environments are not commonly met in commonsense domains. Furthermore, HCDs fall under the agent's control, enabling it to manage their size according to resources.

⁴ Jess: <http://www.jessrules.com/> (last accessed: June 2012)

4 Implementation

The reasoner comprises different modules to facilitate developers in the construction of Event Calculus theories, the refinement of production rules and the monitoring of system execution at runtime. *DEC* and DECKT axiomatizations are implemented on top of Jess. As a consequence, the resulting programs inherit the declarative epistemic and non-epistemic semantics, while the reasoning cycle's operational semantics enable for online reasoning in dynamic environments.

A multitude of features have been integrated to the reasoner in order to enhance the implementation of its operational behavior, while still being consistent with the basic tenets of its axiomatizations. The support for both epistemic and non-epistemic derivations into a production framework, as well as the requirement for runtime execution, led to the introduction of alternative mechanisms to the reasoning process, which are not typically met in the signature of the original formalisms. For instance, instead of implementing parallel circumscription of predicates as employed by standard Event Calculus, negation-as-failure (NaF) and the semi-destructive update of the KB, which are encompassed in our rule-based system, offer a solution to the computational frame problem without the need to write additional frame axioms to perform predicate completion.

Moreover, although the underlying formalisms are neutral with respect to the order of rule execution, the salience value-based conflict resolution strategy used by the rule engine can be used as an additional feature in the operational semantics to handle concurrent event occurrences. The support for reification of fluent formulas in Event Calculus predicates was also deemed necessary, as explained in Section 2, due to the epistemic nature of axioms.

The effective deployment of multi-agent systems calls for tools that can support the developer in all steps of design and implementation cycles through graphical user interfaces. A visual development environment enables the programmer to implement the mental state of rational agents at a more abstract level by either parsing only Event Calculus axiomatizations or in addition by modifying specific rules of the Jess program for specialized tasks. New events and observations can be asserted on-the-fly either manually by means of the user interface or through a Java interface that enables the system to acquire information arriving from sensors and actuators. At a higher level, the user can choose between the execution of classical Event Calculus reasoning or epistemic reasoning, i.e., to incorporate the DECKT axiomatization or not.

Fig. 1 displays the visual development environment of the reasoner through a typical interaction loop: the developer designs a domain axiomatization by means of an intuitive Event Calculus syntax, which is then parsed automatically into appropriate Jess rules and finally, at execution time, the programmer is informed about the progress of reasoning and the elicitation of commonsense knowledge. *DEC* and DECKT domain independent axioms are precompiled as Jess rules. Tuple (a) of Table 1 for instance shows the translation into a production rule of the Event Calculus axiom that captures the positive influence of event occurrences on the state of fluents. According to the rule syntax, variables start with a question mark '?', the ' \leftarrow ' operator performs pattern binding to store a reference to a fact

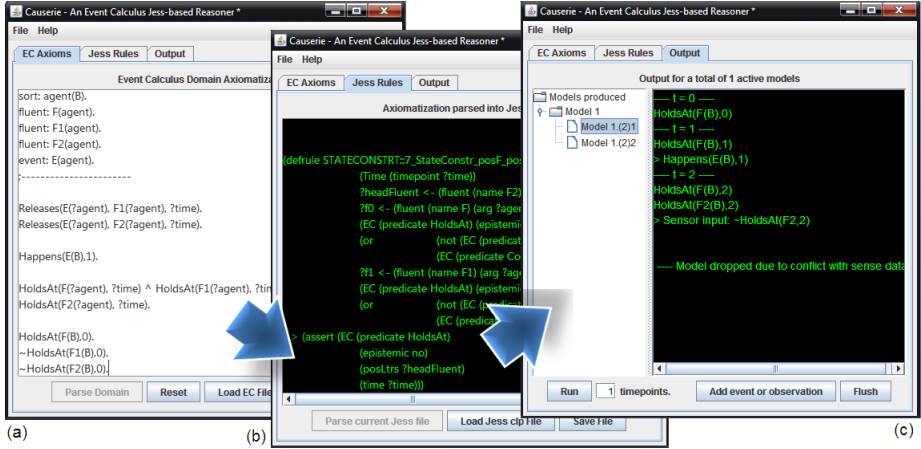


Fig. 1. The user interface facilitates developers in writing EC axioms, parsing them as Jess rules and offering online reasoning functionalities

of the working memory, in a variable, whereas the '\$' symbol denotes lists of items (i.e., fluents in this case). For example, the proposition $HoldsAt(Knows(F1 \vee \neg F2 \vee F3), 3)$ is stored as the fact $(EC \text{ (predicate HoldsAt) (epistemic Knows) (posLtrs F1 F3)(negLtrs F2)(time 3)})$. Conjunctions of epistemic formulas are decomposed into the individual components, as permitted by the properties of knowledge. For non-epistemic fluents the *posLtrs* slot contains exactly one fluent, whereas *negLtrs* is empty.

Example 1. Let the domain description \mathcal{D}_1 for the three fluents $F1, F2, F$ where Σ contains only the effect axiom $HoldsAt(F1, t) \wedge \neg HoldsAt(F2, t) \Rightarrow Initiates(E, F, t)$, and $\Delta_1 = \{Happens(E, 2)\}$, while no non-epistemic fluent is released initially, i.e., $\neg ReleasedAt(f, t) \in I$. At design-time, the Java compiler instantiates all \mathcal{DEC} and \mathcal{DECKT} domain meta-axioms given the particular domain description and produces a set of Jess rules. For example, the instantiation of (KT3.1), (KT6.1.1) \mathcal{DECKT} meta-axioms for the effect axiom of \mathcal{D}_1 appears on Table 1(b) (with some simplifications for readability purposes). At runtime the available knowledge at each timepoint is taken into account in the head of the rule, in order to dynamically reconfigure its behavior. In particular, the body of the rule checks that no precondition is known not to hold, whereas in the head the two lists *?unknownPosF* and *?unknownNegF* collect the unknown precondition fluents to construct the $C(t)^-$ set.

Referring to domain \mathcal{D}_1 , it is possible for the reasoner to show entailments, such as the following (fluents not explicitly mentioned in the initial state are assumed to be false, by application of NaF):

$$\mathcal{D}_1 \wedge HoldsAt(Knows(F1), 3) \wedge HoldsAt(Knows(\neg F2), 3) \models_{\mathcal{DEC}, \mathcal{DECKT}} HoldsAt(Knows(F), 4)$$

Table 2. A trigger axiom with causal and temporal constraints

Event Calculus Axioms	Jess Rules
$\text{Happens}(\text{StartHeating}(\text{Pot}), ?t1) \wedge$ $\sim(\text{Happens}(\text{StopHeating}(\text{Pot}), ?t3) \wedge$ $\{?t3 \geq ?t1\} \wedge \{?t3 < (?t1+5)\})$ \Rightarrow $\text{Happens}(\text{StartBoiling}(\text{Pot}), ?t1+5).$	<pre>(defrule TRIGGERAXIOMS::Trigger_StartBoiling (Time (timepoint ?current_time)) ?e0 <- (event (name StartHeating) (arg Pot)) (EC (predicate Happens) (event ?e0) (time ?t1)) (not (and ?e2 <- (event (name StopHeating) (arg Pot)) (EC (predicate Happens) (event ?e2) (time ?t3)) (test (>= ?t3 ?t1)) (test (< ?t3 (+ ?t1 5)))))) (test (= (+ ?t1 5) ?current_time)) ?event <- (event (name StartBoiling) (arg Pot)) => (assert (EC (predicate Happens) (epistemic no) (event ?event) (time ?current_time))))</pre>

5 An Application Domain

This section illustrates the contribution of causal rule-based reasoning in a real-world application domain. Specifically, we describe how the reasoner forms part of a general framework for context-aware Ambient Assisted Living (AAL) services, in order to monitor user actions, recognize parallel activities, detect hazardous situations and react in a timely manner.

The development of context-aware sensor-rich smart spaces that adapt to user’s preferences and needs is largely based on the research trends of Ubiquitous Computing and Ambient Intelligence. Of the most prominent applications of this research has been the implementation of AAL environments. The development of enabling technologies to support people with mild cognitive disabilities and dementia while they carry out everyday domestic activities is attracting considerable attention. The aim is to increase the autonomy of this population within their own living environment and improve their feeling of safety, a goal that is considered essential both by patients and care providers. The current trend in representing, recognizing and reasoning about human activities in context-aware smart spaces relies largely on Semantic Web languages and reactive systems, which, although mature enough, still do not meet many of the challenges of such complex domains, in terms of expressing and reasoning about complex situations [13]. Whereas rule-based systems are particularly efficient for implementing reactive behavior upon recognized situations, deliberating in ambient domains requires the integration of more powerful cognitive skills, potentially under partial observability of the environment.

To our opinion, commonsense and temporal reasoning are key issue that need to be encapsulated in entities that inhabit smart spaces. Already the Event Calculus is regarded a prominent candidate to deliver solutions for more advanced tasks in such domains [14, 15]. Our intension is to go beyond current implementations of reactive smart spaces in ways that consider both the current knowledge and future contingencies of user activities.

Example 3. (*Ambient Assisted Living*) Let a domain axiomatization where changes in the environment can be due to both user actions or physical events. The user can move between rooms, change the location of objects or manipulate

objects, e.g. turn on the hot plate. A pot will start heating up and eventually its content will start to boil (Table 2) under appropriate conditions:

$$\text{HoldsAt}(\text{TurnedOn}(\text{HotPlate}), t) \wedge \text{HoldsAt}(\text{PlacedOn}(\text{obj}, \text{HotPlate}), t) \wedge \neg \text{HoldsAt}(\text{Heating}(\text{obj}), t) \Rightarrow \text{Happens}(\text{StartHeating}(\text{obj}), t)$$

The monitoring system can identify potentially critical situations both about the present and the future, in order to trigger different types of alerts and recommendations in an as less intrusive manner as possible. For example, it can foresee that the water will boil after a certain period of time, postponing any alert action until a later point. Table 2 shows a sample trigger axiom with temporal constraints and its corresponding Jess rule. If in the meantime the user turns on the bathtub faucet and starts filling the bathtub with water, the system should identify that these two parallel activities will demand the user’s attention at approximately the same time at two different locations, i.e., she should stop the water from reaching the rim of the bathtub while also turn off the hot plate in the kitchen. Although the critical situation refers to a future point in time and it is not certain that it will actually occur, a reminder is more appropriate to be placed in the present state. In certain cases the system may need to take initiative, such as to turn off the hot plate for her. \square

Context knowledge in the general framework for AAL is captured in OWL ontologies and translated according to the reasoner’s syntax, in order to formalize the causal properties of the domain and enable commonsense reasoning. For instance, the open world assumption adopted by OWL is directly accommodated by the epistemic component of the reasoner by application of NaF on epistemic fluents: unless explicitly stated, fluents are by default unknown. Furthermore, the representation of subsumption relations among domain objects is supported by the reasoner in the definition of fluents or events. The following two sorts describe classes and instances for the domain of Example 3 and characterize objects that exist in the kitchen:

sort : *kitchenware*(*Toaster, Kettle, Pot, Stove, Table*).

sort : *movable_kitchenware*(*Toaster, Kettle, Pot*).

These sorts reflect the fact that *movable_kitchenware* \sqsubseteq *kitchenware* as obtained by the domain ontology. As such, the following event definition restricts the range of permitted values when grounding its arguments:

event : *PutOn*(*actor, movable_kitchenware, kitchenware*).

Given a narrative of actions performed by the user and observations obtained from the environment, the non-epistemic component plays a dual part in the framework: it performs high-level activity and situation recognition of the current state and it progresses the KB into the future to identify exceptional situations, to activate alerts and to determine the most appropriate time to trigger them. The practical significance of elaborating on the feasibility or fidelity of a course of actions by performing temporal projection is evidenced in recent implementations of autonomous systems (e.g. [16]). Epistemic reasoning can be applied in case of partial observability of world aspects, e.g., when mobile devices such

as robots or PDAs need to perform lightweight reasoning tasks having limited access to the full description of the KB due to connection failures or resource constraints. Situations that require epistemic reasoning in Ambient Intelligence environments are described in [17]. Given the knowledge at hand, the epistemic component of the reasoner can create a partial description of the world state by associating and maintaining causal relations among unknown world parameters that change dynamically. The causal relations provide a valuable source of information, not only for outlining an agent's epistemic reflection of the world state, but also for the determination of which sense actions to perform and when. A powerful epistemic model can enhance significantly the cognitive skills of deliberating agents; even parameters not directly measurable may be indirectly inferred when studying the epistemic ramifications of actions.

6 Discussion and Conclusion

The Event Calculus differentiates from other action theories due to its ability to perform causal and temporal reasoning for a wide range of commonsense phenomena. Nevertheless, its main use has been focused on deductive state progression given a known narrative of actions or abductive planning by application of backward chaining strategies. Its application to dynamic environments, which is the formalism's primary target domain that calls for forward chaining methods, has only recently started to attract attention [7, 15]. As a result, a rule-based implementation of the Event Calculus, as presented in the present study, achieves to exploit the rich expressiveness of the formal languages with the benefits of rule-based theories for online reasoning. The systems presented in [7] and [18], sharing a similar objective, focus furthermore on combining both forward- and backward-chaining rules and formally prove the properties of the operational semantics. On the other hand, their capacity in modeling complex phenomena, such as partial observability of the world state or non-determinism, is limited. Our current system manages to encompass solutions for supporting reification of fluents, multiple model generation and others.

Currently, the implementation of the epistemic component follows the progress made in the theoretical foundations of DECKT. As a result, there are limitations in the range of domains that can be modeled to perform epistemic reasoning and not all non-epistemic axiomatizations can fully be expressed as epistemic theories. For instance, according to the formal definition for knowledge, sensed information cannot contradict already established knowledge. The extension of DECKT to apply belief revision strategies is our next logical step. Moreover, temporal indeterminacy is not formalized in DECKT; therefore, actions happening at some timepoint within an interval are not permitted for epistemic reasoning. On the other hand, potential occurrences of events are formally axiomatized, i.e., when the preconditions are unknown as in Example 3. Complex benchmark domains have been modeled under partial observability to express commonsense phenomena, such as complex ramifications, delayed knowledge effects, potential event occurrences and others (e.g., Shanahan's circuit [19]).

This work provides the substrate for both theoretical and practical future work. It offers the guidelines for the implementation and evaluation of different versions of epistemic and non-epistemic reasoning. Apart from belief revision, an interesting extension is to permit the attachment of certainty weights to obtained models, in order to accommodate approaches that couple symbolic and quantitative reasoning. From the practical standpoint, we work on the evaluation the reasoner's expressive capacity and performance using benchmark problems in comparison with SAT or ASP implementations of the Event Calculus, as well as under real-world conditions within our AAL framework.

References

1. Levesque, H., Reiter, R.: High-level Robotic Control: Beyond Planning. A Position Paper. In: AIII 1998 Spring Symposium: Integrating Robotics Research: Taking the Next Big Leap (1998)
2. Van Harmelen, F., Lifschitz, V., Porter, B.: Handbook of Knowledge Representation. Elsevier Science, San Diego (2007)
3. Scherl, R.B., Levesque, H.J.: Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1-2), 1–39 (2003)
4. Thielscher, M.: Representing the Knowledge of a Robot. In: KR 2000: Principles of Knowledge Representation and Reasoning, pp. 109–120 (2000)
5. Lobo, J., Mendez, G., Taylor, S.R.: Knowledge and the action description language A. *Theory and Practice of Logic Programming* 1(2), 129–184 (2001)
6. Patkos, T., Plexousakis, D.: Reasoning with Knowledge, Action and Time in Dynamic and Uncertain Domains. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, pp. 885–890 (2009)
7. Chesani, F., Mello, P., Montali, M., Torroni, P.: A logic-based, reactive calculus of events. *Fundamenta Informaticae* 105, 135–161 (2010)
8. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. *New Generation Computing* 4(1), 67–95 (1986)
9. Mueller, E.: *Commonsense Reasoning*, 1st edn. Morgan Kaufmann (2006)
10. Patkos, T., Plexousakis, D.: Efficient epistemic reasoning in partially observable dynamic domains using hidden causal dependencies. In: NRAC 2011 - 9th International Workshop on Non-Monotonic Reasoning, Action and Change, pp. 55–62 (2011)
11. Chittaro, L., Montanari, A.: Efficient Temporal Reasoning in the Cached Event Calculus. *Computational Intelligence* 12, 359–382 (1996)
12. Dimopoulos, Y., Kakas, A.C., Michael, L.: Reasoning About Actions and Change in Answer Set Programming. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 61–73. Springer, Heidelberg (2004)
13. Riboni, D., Bettini, C.: Owl 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing* 7, 379–395 (2011)
14. Chen, L., Nugent, C., Mulvenna, M., Finlay, D., Hong, X., Poland, M.: A Logical Framework for Behaviour Reasoning and Assistance in a Smart Home. *International Journal of Assistive Robotics and Mechatronics* 9 (2008)
15. Sadri, F.: Intention recognition with event calculus graphs. In: *Web Intelligence and Intelligent Agent Technology*, pp. 386–391 (2010)

16. Kunze, L., Dolha, M.E., Beetz, M.: Logic programming with simulation-based temporal projection for everyday robot object manipulation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011), pp. 3172–3178 (2011)
17. Patkos, T., Plexousakis, D.: Epistemic reasoning for ambient intelligence. In: 3rd International Conference on Agents and Artificial Intelligence (ICAART), pp. 242–248 (2011)
18. Kowalski, R., Sadri, F.: An Agent Language with Destructive Assignment and Model-Theoretic Semantics. In: Dix, J., Leite, J., Governatori, G., Jamroga, W. (eds.) CLIMA XI. LNCS, vol. 6245, pp. 200–218. Springer, Heidelberg (2010)
19. Shanahan, M.: The ramification problem in the event calculus. In: 16th International Joint Conference on Artificial Intelligence (IJCAI), pp. 140–146 (1999)

On the Algebraic Semantics of Reactive Rules^{*}

Katerina Ksytra, Nikolaos Triantafyllou, and Petros Stefanias

National Technical University of Athens,
Iroon Polytechniou 9, 15780 Zografou, Athens, Greece
{katksy,nitriant}@central.ntua.gr,
petros@math.ntua.gr

Abstract. Service oriented architectures and event driven environments are becoming dominant over the web. Reactive Rules expressed by Rule Markup Languages are used to define the system's reactions. In this paper we present a Hidden (Sorted) Algebra approach to some of the most common families of Reactive Rules. This semantics will allow the mapping between Rule Markup Languages and Behavioral Algebraic Specification Languages. Verification techniques for reactive rules, will provide automated reasoning capabilities and support the development of new rule based policies and trust models.

Keywords: Reactive Rules, Hidden Algebra, Observational Transition Systems, Formal Semantics.

1 Introduction

The study of Reactive Rules for event driven applications started to become extensive during the 1990s. The interest was on rules that specify the behavior of systems that trigger actions as response to the detection of events from the environment. As the authors of [10] point out, today's research on Event Driven Architecture IT infrastructures like on-Demand or Utility Computing, Real-Time Enterprise, Business Activity Management and so on, is intensive. In addition, a new push has been given in the field due to the strong demand of the web community for event processing functionalities for Semantic Web Markup Rule Languages such as RuleML.

Over the years different approaches to reactive event processing were developed. As analyzed in [10], in active databases the focus is on the support of automatic triggering of global rules in response to either internal updates or external events. In event notification and messaging systems, the focus is on the sequence of events in a given context, while in event/action logics it is on the

^{*} This paper was supported by the THALIS project "Algebraic Modeling of Topological and Computational Structures and Applications". The Project THALIS is implemented under the Operational Project Education and Life Long Learning and is co-funded by the European Union (European Social Fund) and National Resources (ESPA).

inferences that can be made from the fact that certain events have occurred. For a survey on the event/action/state processing space we refer the interested reader to [1]. Some of the most important families of reactive rules used in the semantic web community are Production rules, Event Condition Action (ECA) rules and Knowledge Representation (KR) rules. Also great interest is shown for the definition of Complex Events.

We will present Observational Transition System (OTS) semantics for some rule families, as depicted in figure 1. While there are many approaches in terms of expressing the meaning of reaction rules, our contribution focuses on the foundations for creating libraries of rules with verified behavior and allows the composition of these rules in order to create more complex rule bases that preserve desired properties. This can be achieved due to the strong modularisation properties of hidden algebra, such as information hiding, renaming and sum [13]. Hidden algebra has been successfully applied to system’s design verification. By expressing reaction rules in the same framework we could reason not only about the rules but also on their behavior in particular systems. Our paper is organized as follows. In the second section, we briefly introduce the basic concepts we will need. In section 3 we give OTS semantics for some of the most commonly used types of reactive rules, and present some case studies.

2 Preliminaries

2.1 Hidden Algebra

Hidden Algebra is an approach for giving semantics to concurrent distributed object oriented systems, as well as to software systems in general. It is a version of behavioral type in the object oriented paradigm [3], called behavioral specification.

For a set of sorts S , we say that the set A is S -sorted if it can be regarded as a family of sub-sets as $A = \cup\{A_s\}_{s \in S}$. Using this set of sorts, we can define a signature Σ as the pair $\langle S, F \rangle$ where F is a set of function symbols. Such that F is equipped with a mapping $F \rightarrow S^* \times S$ meaning that each $f \in F$, $f : s_1 \times \dots \times s_n \rightarrow s$. Then the type (or rank) of f is defined as $rank(f) = s_1 \dots s_n s \in S^*$. Given a signature as above, a Σ -Algebra A consists of a non-empty family of

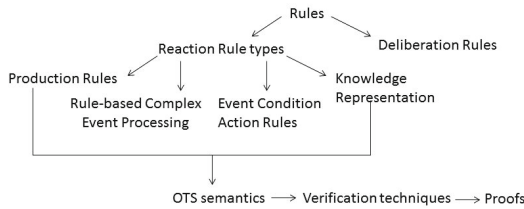


Fig. 1. Reaction rules families and OTS

carrier sets $\{A_s\}_{s \in S}$ and a total function $f^A : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ for each function symbol $f : s_1 \times \dots \times s_n \rightarrow s \in F$.

A Σ -homomorphism between two Σ -algebras A and B, denoted by $h : A \rightarrow B$, is a family of maps $\{h_s : A_s \rightarrow B_s\}_{s \in S}$ that preserves the operators. By imposing a partial ordering on the sorts we get an Order Sorted Σ -Algebra (OSA).

An *order sorted signature* is a triple (S, \leq, Σ) such that (S, Σ) is a many-sorted signature, (S, \leq) is a poset, and the operators satisfy the following monotonicity condition; $\Sigma \in \Sigma_{w_1, s_1} \cap \Sigma_{w_2, s_2}$ and $w_1 \leq w_2$ implies $s_1 \leq s_2$. Given a many-sorted signature, a (S, Σ) -algebra A is a family of sets $\{A_s \mid s \in S\}$ called the carriers of A, together with a function $A_\Sigma : w \rightarrow A_s$ for each Σ in $\Sigma_{w, s}$. Where $A_w = A_{s_1} \times \dots \times A_{s_n}$ and $w = s_1 \dots s_n$. Let (S, \leq, Σ) be an order sorted signature. A (S, \leq, Σ) -algebra is a (S, Σ) -algebra A such that, $s \leq s'$ in S implies $A_s \subseteq A_{s'}$ and $\Sigma \in \Sigma_{w_1, s_1} \cap \Sigma_{w_2, s_2}$ and $w_1 \leq w_2$ implies $A_\Sigma : w_1 \rightarrow A_{s_1}$ equals $A_\Sigma : w_2 \rightarrow A_{s_2}$ on A_{w_1} [8]. The purpose of the formalisation of order sorted signatures is to define sorts (similar to classes in OO), functions (similar to methods in OO) and inheritance between the sorts.

In Hidden Algebra [3] two kinds of sorts exist: visible sorts and hidden sorts. Visible sorts represent the data part of a specification while hidden sorts denote the state of an abstract machine.

Given a signature (S, \leq, Σ) and a subset $H \subset S$ the hidden sorts, a hidden algebra (or a hidden model in general) A interprets the visible sorts V and the operations Ψ of the visible sorts as a fixed model D (the data model, say an order sorted algebra) such that $A \upharpoonright_{V, \Psi} = D$ (where \upharpoonright is the model reduct). Given two signatures (S, \leq, Σ) and (S', \leq, Σ') a signature morphism $\phi : (S, \leq, \Sigma) \rightarrow (S', \leq, \Sigma')$ consists of a mapping z on sorts that preserves the partial ordering, i.e. for $s \leq s'$ then $z(s) \leq z(s')$ and an indexed mapping on operators g , such that $\{g_{s_1 \dots s_n} : \Sigma_{s_1 \dots s_n, s} \rightarrow \Sigma'_{z(s_1) \dots z(s_n), f(s)}\}_{s_1 \dots s_n, s \in S, n \geq 0}$. We will refer to operators whose arguments contain a hidden sort and/or whose returned value is a hidden sort, as hidden or behavioral operators.

The hidden signature morphism $\phi : (S, H, \leq, \Sigma) \rightarrow (S', H', \leq, \Sigma')$ preserves the visible and hidden part of the signatures and obeys to the following conditions: (a) g maps each behavioral operator to a behavioral operator, (b) if $z(h) < z(h')$ for arbitrary visible sorts h and h' then $h < h'$ and (c) if $\sigma' \in \Sigma_{w', s'}$ is a behavioral operator where $w \in (S \cup H)^*$ and some sort of w is hidden, then $\sigma' = g(\sigma)$ for some behavioral operator $\sigma \in \Sigma$.

2.2 Observational Transition Systems

An Observational Transition System (OTS) is a transition system that can be written in terms of equations and is a proper sub-class of behavioral specifications. Assuming there exists a universal state space Y and that each data type we need to use is already defined in terms of an initial algebra, an OTS S is defined as the triplet $S = \langle O, I, T \rangle$, where [4]:

1. O is a finite set of observers. Each $o \in O$ is a function $o : Y \rightarrow D$, where D is a data type (visible sorted) and may differ from observer to observer. Given an OTS S and two states $u_1, u_2 \in Y$ the equivalence between them with respect to the OTS S is defined as $\forall o \in O, o(u_1) =_S o(u_2)$.
2. I is the set of initial states of the system such that $I \subseteq Y$.
3. T is a finite set of transition functions. Each $\tau \in T$ is a function $\tau : Y \rightarrow Y$, such that $\tau(u_1) = \tau(u_2)$ for each $[u] \in Y/_=S$ and $u_1, u_2 \in [u]$. The state attained by $\tau(u)$ is called the successor state of u with respect to S . Also with each τ comes a condition $c\text{-}\tau$, called the effective condition of τ , such that $\tau(u) =_S u$ if $\neg (c\text{-}\tau(u))$.

Observers and transitions are usually parameterized and generally they are denoted as $o_{i_1 \dots i_m}$ and $\tau_{j_1 \dots j_n}$ provided that there exist data types $D_k, k \in \{i_1 \dots i_m, j_1 \dots j_n\}$ and $m, n \geq 0$.

2.3 Timed Observational Transition Systems

Timed observational transition systems, TOTSSs, are OTSSs that are evolved by introducing clock observers in order to deal with timing. Again Y denotes a universal state space. Let \mathbb{B}, \mathbb{N} and \mathbb{R}^+ be a set of truth values, a set of natural numbers and a set of non-negative real numbers, respectively. A TOTSS $S = \langle O, I, T \cup \{tick_r \mid r \in \mathbb{R}^+\} \rangle$ where [9]:

1. O is a set of observers. The set $O = D \cup C$ is classified into the set D of discrete observers and the set C of clock observers. Clock observers may be called clocks. The discrete observers are defined in the same way as the observers of an OTS.
2. I is the set of initial states such that $I \subseteq Y$.
3. $T \cup \{tick_r \mid r \in \mathbb{R}^+\}$ is a set of conditional transitions. Each $\tau \in T \cup \{tick_r \mid r \in \mathbb{R}^+\}$ is a function $\tau : Y \rightarrow Y$.

For each clock observer $o \in C$ where $o : Y \rightarrow D$, D is a subset (subtype) of $\mathbb{R}^+ \cup \{\infty\}$. For each $\tau \in T$, there are two clocks $l_\tau : Y \rightarrow \mathbb{R}^+$ and $u_\tau : Y \rightarrow \{\mathbb{R}^+\} \cup \{\infty\}$, which return the lower and upper bounds of τ , respectively. They are basically used to force τ to be executed, or applied between the lower bound returned by l_τ and the upper bound returned by u_τ . There is also one special clock $now : Y \rightarrow \mathbb{R}^+$. It serves as the master clock and returns the time amount that has passed after starting the execution of S . now initially returns 0. C contains the two clocks l_τ and u_τ for each $\tau \in T$, and the master clock now . For each $\tau \in T$, its effective condition consists of the timing part and the non-timing part. The non-timing part is denoted by c_τ . Given a state $u \in Y$, the timing effective condition is $l_\tau \leq now(u)$. Each $tick_r$ is a time advancing transition. Given a state $u \in Y$, for each $tick_r$, its effective condition is $now(u) + r \leq u_\tau(u)$ for each $\tau \in T$, and $now(tick_r(u))$ is $now(u) + r$ if the effective condition of $tick_r$ is true in u .

3 Reactive Rules and Observational Transition Systems

3.1 Production Rules and OTSs

Our aim is to provide an OTS semantics to production rules. Production rule systems use a set of condition-action rules cyclically invoking (assert, retract, etc.) actions when tests over their working memory succeed. Their syntax is: *if* C_i *do* A_i , where A_i denotes an action that must be applied automatically by the system when it detects that the conditions denoted as C_i hold.

The semantics of these actions A_i could be those of transition rules in an OTS, because both transition rules and actions A_i cause explicit changes to the state of the system when they are applied. The conditions defined above can be mapped to (or be part of) the effective conditions of the transition rules.

So at first, it looks like the OTS semantics of production rules are quite straightforward. But a closer look will reveal that there is an error in the previous reasoning. In the OTS approach we can define under which conditions the transitions will be successful but it is not possible to enforce the application of those transitions on a state. On the other hand the production rules system must react in the desired way when the conditions are met. So the semantics of production rules in OTS should be the enforcement of the application of a transition rule in a system state.

Here, we will attempt to give such semantics by providing a typing on the states of an OTS. Then we will define an OTS that contains only transition rules from the typed states, which can be regarded as a sort of typed OTS.

As mentioned in section 2.2, an OTS corresponds to an order sorted hidden algebra (S, \leq, Σ) . We will use the hidden sorts, defined as $H \subset S$, to produce the typing system. For the following we assume a set of hidden sorts H , denoting the state space of the OTS Y , meaning that $\forall u \in Y, \exists h \in H$ such that the sort of u belongs to the sort h .

Definition 1. *Suppose that the OTS contains n -transition rules $\tau_i : H D_{1_i} \dots D_{n_i} \rightarrow H$, $i \in (1, \dots, N)$ and that each transition rule is affected by the effective condition $c\text{-}\tau_i : H D_{1_i} \dots D_{n_i} \rightarrow \text{Bool}$. For each transition τ_i and for all visible sort constants $d_{k_i} \in D_{k_i}$ such that there $\exists u \in Y$ for which $c\text{-}\tau_i(u, d_{1_i}, \dots, d_{n_i}) = \text{true}$ we define;*

- a hidden sub-sort, $h_{id_{1_i} \dots d_{n_i}} \leq H$. Such that $\forall u \in h_{id_{1_i} \dots d_{n_i}}$ we have that $c\text{-}\tau_i(u, d_{1_i}, \dots, d_{n_i}) = \text{true}$. This is the hidden sub-sort that satisfies the effective condition of the transition rule τ_i for visible sort arguments d_{1_i}, \dots, d_{n_i} .
- also for each transition rule $\tau_i : H D_{1_i} \dots D_{n_i} \rightarrow H$ we define a new transition $\tau'_{id_{1_i} \dots d_{n_i}} : h_{id_{1_i} \dots d_{n_i}} \rightarrow H$, such that $\tau'_i(u) = \tau_i(u, d_{1_i}, \dots, d_{n_i}) \forall u \in h_{id_{1_i} \dots d_{n_i}}$.

We will call the OTS $S' = \langle O', I', T' \rangle$ defined from $S = \langle O, I, T \rangle$ as

- $O' = O$
- $T' = \{\tau'_i \mid \tau_i \in T\}$
- $I' = I$

the production OTS of S .

Now for a given state of the system $u \in Y$ it is possible to decide if it belongs to a sort or not, by checking whether $\exists d_{i_k}$ such that $c\text{-}\tau_i(u, d_{i_1}, \dots, d_{i_n}) = \text{true}$. If the previous expression holds then $u \in h_{id_{i_1} \dots d_{i_n}}$. The next definition describes how with the above formalism we can semantically interpret a set of production rules as an OTS.

Definition 2. Assume a set of production rules: if C_i do A_i , $i \in \{1, \dots, N\}$. In such a set of rules C_i defines a set of constraints that when they hold the system should automatically apply the action(s) A_i . We define an OTS $S = \langle O, I, T \rangle$ such that the actions A_i 's of the above rules are mapped to some transition rules τ_i 's and the conditions C_i 's to effective conditions $c\text{-}\tau_i$'s. The production OTS $S' = \langle O', I', T' \rangle$, created from S using definition 1 is the semantic interpretation of the production rules in the OTS framework.

Example 1. As an example of the previous definition consider the following production rule for an online store: "If status of client is premium and the type of product is regular then assert discount of 25 percent for the customer" [2]. It possible to fully characterize the above system with a set of queries (observers) as those in table 1. Where Client, Status, Product, Percent and Type are predefined visible sorts. We map the assert action to a transition rule, $\text{Assert} : H \text{ Client Product} \rightarrow H$. Given a client C and a product P, the effective condition for this transition rule is defined by the signature $c\text{-Assert} : H \text{ Client Product} \rightarrow \text{Bool}$ and the equation $c\text{-Assert}(H, C, P) = (\text{client-status}(H, C) = \text{premium}) \wedge (\text{product-type}(H, P) = \text{regular})$.

Now for each pair of constants c_i and p_i such that there exists a system state u that makes $c\text{-Assert}(u, c_i, p_i) = \text{true}$, we define a new subsort $h_{c_i p_i} \leq H$. Finally for each such subsort we define a transition rule, $\text{ASSERT}_i : h_{c_i p_i} \rightarrow H$, such that $\forall u \in h_{c_i p_i}$ we have; $\text{Assert}_i(u) = \text{Assert}(u, c_i, p_i)$. These transition rules can only be applied in states $u \in h_{c_i p_i}$, and in those states they are the only applicable transitions. So basically in an OTS containing only these transition rules we are enforcing the application of the desired transitions.

Following definition 2 we define the OTS $S' = \langle O', I', T' \rangle$ that corresponds to the production rule as follows:

- $O' = \{ \text{client-status, product-type, discount} \}$
- I' the set of initial states
- $T' = \{ \text{Assert}_i \mid \text{such that } h_{c_i p_i} \leq H, \}$

Table 1. Observers of an OTS specifying a client discount system

Observer	Signature	Informal Definition
client-status	$H \text{ Client} \rightarrow \text{Status}$	returns the status of the client, $\{\text{premium}, \neg\text{premium}\}$
product-type	$H \text{ Product} \rightarrow \text{Type}$	returns the type of the product, $\{\text{regular}, \neg\text{regular}\}$
discount	$H \text{ Client Product} \rightarrow \text{Percent}$	returns the discount the client has on a product

In languages that implement OTS specifications definition 2 can not be easily applied (possible infinity of sub-sorts). For this reason we will present a specification approach to the semantics of production rules and prove that the two OTSs, S and S' defining the state spaces Y and Y' respectively, are behaviorally equivalent¹ [15].

In an OTS, a state is a kind of black box. This means that each state is characterized only by the observable values returned by the observers $o \in O$. As a result, the effect of a transition rule (or a change of the state in general) can be characterized by the values returned by these observers. Now assuming a set of production rules of the form if C_i do A_i , with $i \in \{1, \dots, N\}$, we would ideally wish to correspond action A_i to a transition rule τ_i with an effective condition C_i . As we argued before it is not possible to enforce the application of a transition rule in an OTS. On the other hand it is possible to have conditional observations. So for an arbitrary state $u \in Y$ we have that after the application of the transition rule τ_i , $o_1(\tau_i(u, d_1, \dots, d_n), v_1, \dots, v_k) = v_1$ if $C_i(u, d_1, \dots, d_n)$. These observations, led to the following definition.

Definition 3. We define an OTS $S = \langle O, I, T \rangle$ from a set of production rules if C_i do A_i , with $i \in \{1, \dots, N\}$ as:

- $T = \{\text{read}\}$, i.e. the only transition rule is $\text{read} : H \rightarrow H$, a single transition with no input.
- I is a set of initial states, $I \subseteq Y$.
- $O = \{o_1, \dots, o_k\}$ a finite set of observers. Each observer $o_i : HV_{i_1} \dots V_{i_n} D_{i_1} \dots D_{i_k} \rightarrow H$ satisfies the following equations for an arbitrary system state u :
 - $o_i(\text{read}(u), v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k}) = v_1$ if $C_1(u, d_{i_1}, \dots, d_{i_k})$
 - $o_i(\text{read}(u), v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k}) = v_2$ if $C_2(u, d_{i_1}, \dots, d_{i_k})$
 - ...
 - $o_i(\text{read}(u), v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k}) = v_N$ if $C_N(u, d_{i_1}, \dots, d_{i_k})$

Also $\forall d_{i_1}, \dots, d_{i_k}, d'_{i_1}, \dots, d'_{i_k}$ we have that $\forall o_i \in O, u \in I$ $o_i(u, v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k}) = o_i(u, v_{i_1}, \dots, v_{i_n}, d'_{i_1}, \dots, d'_{i_k})$. This means that initially the values returned by the observers are only depended on the visible sorts v_{i_1}, \dots, v_{i_n} and not on the extra arguments d_{i_1}, \dots, d_{i_k} that are added to allow us to reason about the effective conditions of the transitions. Finally,

$$o_i(\text{read}(u), v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k}) = o_i(u, v_{i_1}, \dots, v_{i_n}, d_{i_1}, \dots, d_{i_k})$$

if $\neg(C_1(u, d_{i_1}, \dots, d_{i_k}) \vee \dots \vee C_N(u, d_{i_1}, \dots, d_{i_k}))$.

The intuition behind the previous definition is that we have a system that has one transition only. This transition basically checks to see if any of the conditions defined by the production rules are met and if so it automatically changes the

¹ We will show that if S and S' have the same set of initial states and the same set of observers O then for $u \in Y$ and $u' \in Y'$ such that $\forall o \in O, o(u, y_1, \dots, y_n) = o(u', y_1, \dots, y_n)$ applying a transition rule τ of S and τ' of S' will result in states $\tau(u), \tau'(u')$ such that $o(\tau(u), y_1, \dots, y_n) = o(\tau'(u'), y_1, \dots, y_n) \forall o \in O$.

values returned by the observers to those we would expect if the corresponding to the condition action was applied.

Proposition 1. *Two OTSs defined from a set of production rules R , using definitions 2 and 3 that have the same set of initial states are behaviorally equivalent, according to observational equivalence.*

Proof. The proof can be found at <http://cafeobjntua.wordpress.com/>

3.2 Event Condition Action (ECA) Rules and OTSs

Event Condition Action rules (ECA) are one of the most commonly used categories of reactive rules. Their syntax is: *on Event if Condition do Action*, where event denotes an explicit action that changes the state of the system, and action denotes a change in the state of the system that is caused as a reaction to the event, if the condition part of the rule holds.

Definition 4. *In OTS notation the concept of an ECA-rule, $r = \text{on } E_i \text{ if } C_i \text{ do } A_i$, can be transferred naturally. Suppose transition rules $\tau : S D_1 \dots D_n \rightarrow S$ and $\tau' : S D'_1 \dots D'_k \rightarrow S$, that are under the effective conditions $c-\tau$ and $c-\tau'$ respectively. We assume that the first rule specifies the state change due to E_i and the second rule specifies the state change due to A_i . We also assume that $c-\tau'$ corresponds to C_i . We map r to a new transition rule in the OTS $r : S D_1 \dots D_n D'_1 \dots D'_k \rightarrow S$. Such that for an arbitrary state u and for all observers o that change their returned value when τ' is applied we have: if $o(\tau'(u, d'_1, \dots, d'_k), v_1, \dots, v_m) = v_o$ when $c-\tau'(u, d'_1, \dots, d'_k)$, then $o(r(u, d_1, \dots, d_n, d'_1, \dots, d'_k), v_1, \dots, v_m) = v_o$ when $c-\tau'(\tau(u, d_1, \dots, d_n), d'_1, \dots, d'_k)$ and $c-\tau(u, d_1, \dots, d_n)$. For all observers o' whose observations remain unaffected by the transition rule τ' we define; if $o'(\tau(u, d_1, \dots, d_n), v'_1, \dots, v'_q) = v'_o$ then $o'(r(u, d_1, \dots, d_n, d'_1, \dots, d'_k), v_1, \dots, v_m) = v'_o$ if $c-\tau'(\tau(u, d_1, \dots, d_n), d'_1, \dots, d'_k)$ and $c-\tau(u, d_1, \dots, d_n)$. This transition rule r , basically defines the sequential application of transition rules τ and τ' .*

Example 2. As an example assume the following ECA rule: "On receiving premium notification from marketing and if regular derivable do send discount to customer" [2]. We identify the following components in that rule: The event "receiving premium notification from marketing", the condition "regular derivable" and the action "send discount to customer". By corresponding the event and action to transitions and using the appropriate observers we can define an OTS S as shown on tables 2 and 3.

According to definition 4, we can use the previous OTS to define an OTS $S' = \langle O', T', I' \rangle$ that models the ECA rule as:

- $I' = I$, where I is the set of initial states of S
- $O' = O$, where $O = \{customer - type, discount, product - type\}$
- $T' = \{t\}$, where $t : S \text{ SenderID ClientID ProductID} \rightarrow S$, represents the sequential application of the transitions *receive - premium - notification* and *send - discount*.

The result of applying t to an arbitrary system state S is defined by the following three equations:

1. $discount(t(S, I1, C1, P1), P2, C2) = \text{true}$ if $c\text{-receive-premium-notification}(S, I1, C1) \wedge product\text{-type}(\text{receive-premium-notification}(S, I1, C1), P1) = \text{regular} \wedge (C1 = C2) \wedge (P1 = P2)$
2. $customer\text{-type}(t(S, I1, C1, P1), C2) = customer\text{-type}(\text{receive-premium-notification}((S, I1, C1), C2))$
3. $product\text{-type}(t(S, I1, C1, P1), P2) = product\text{-type}(S, P2)$

In the previous equations $I1$ is a variable denoting an arbitrary sender id, $C1$, $C2$ are variables denoting arbitrary client ids and finally $P1$, $P2$ are variables denoting arbitrary product ids.

Note that on the above definition we define a transition rule that contains, on the effective condition, a reference to a successor state of the arbitrary system state u , namely $\tau(u, d_1, \dots, d_n)$. This approach while providing clear and intuitively straight semantics for ECA rules cannot be applied to the algebraic specification languages that implement OTSs. The reason for this is that it is not permitted to have a transition rule on the right hand side of an equation defining another transition rule. This guarantees the termination of the rewriting procedure that is used to create proofs with such languages. So once again we will define another (semantically equivalent) model for the ECA rules that is more specification orientated than intuitively straight forward. We wish to have transitions (the events) that occur from an outside source to the system like the typical transitions of an OTS, but at the same time we require for our system to react to these events if some conditions hold. We try to achieve this double goal by allowing the systems refresh transition (read) that we defined for the production rules to be parameterized. Also we modify the OTS, with a memory observer, so that it remembers if in the previous state an event occurred or not. The parameterization allows us to simulate the execution of events, while the memory allows us to decide if the OTS must react to this refresh or treat it as an incoming event.

Assume a finite set of ECA-rules $\{r_i = \text{on } E_i \text{ if } C_i \text{ do } A_i \mid i \in \{1, \dots, n \in \mathbb{N}\}\}$ and without harm of generality assume that for $i \neq j; E_i, A_i \neq E_j, A_j$ respectively. Also assume that for these events and actions there exist predefined transition rules in an OTS say S and that the visible sorts D_1, \dots, D_l were required for their definition.

Table 2. Transitions of an OTS specifying a notification client discount system

Transitions	Signature	Informal Definition
receive-premium-notification	$S \text{ SenderID ClientID} \rightarrow S$	transition modeling that a premium notification for ClientID has been sent by SenderID
send-discount	$S \text{ ProductID ClientID} \rightarrow S$	transition that models that a discount for ClientID on ProductID is granted.

Table 3. Observers of an OTS specifying a notification client discount system

Observers	Signature	Informal Definition
customer-type	$S ClientID \rightarrow Status$	observer that returns the status of the given client
discount	$S ProductID ClientID \rightarrow Bool$	returns true (false) if the customer has (no) discount on the product
product-type	$S ProductID \rightarrow Type$	returns the type of the product (regular or not)

Definition 5. Now we define a new OTS modeling these rules, $S' = \langle O', I', T' \rangle$ where:

- $O' = O \cup \{memory\}$. Memory is a special observer that remembers if an event has occurred in a state and what that event was. Since the set of rules is finite we have a finite set of events. We can now define the observer $memory : S D_1 \dots D_l \rightarrow \{1, \dots, n \in \mathbb{N}\}$.
- $T' = \{read\}$. Where $read : S \{1, \dots, n \in \mathbb{N}\} \rightarrow S$, a single parameterized transition function. This according to the value of the index $n \in \mathbb{N}$ models the transition that corresponds to event E_n . For an arbitrary system state u and $i \in \{1, \dots, n \in \mathbb{N}\}$ we define that $\forall o \in O: o(read(u, i), d_1, \dots, d_l, v_1, \dots, v_q) = v_{E_i}$ if $c-\tau_i(u, d_1, \dots, d_l) \wedge (memory(u, d_1, \dots, d_l) = null)$. Here we state that o will return the same value as it would in S , when the transition (event) E_i had occurred successfully, if the memory is empty. The memory is empty at the initial states and after the occurrence of an action.
 Now in the case where the memory is not empty we specify that this triggers a reaction from the OTS with the following equation: $o(read(u, i), d_1, \dots, d_l, v_1, \dots, v_q) = v_{A_i}$ if $c-\tau_{A_i}(u, d_1, \dots, d_l) \wedge (memory(u, d_1, \dots, d_l) = i)$. Here we state that o will return the same value as it would in S , when the transition (action) A_i had occurred successfully, if the memory contains the index i (i.e. in the previous state of S we had an occurrence of the event i).
- * $I' = I$

Revisiting example 2 we can define the OTS $S' = \langle O', I', T' \rangle$, based on definition 5, where:

- $O' = \{customer - type, discount, product - type, memory\}$
- $T' = \{read\}$
- $I' = I$

Since in this example we have one event only, we map it to index 1. So for an arbitrary system state S the effect of $read(S,1)$ on S is defined by the values returned by the observers given in the following equations:

- $customer\text{-}type(read(S,1), C1) = premium$ if $c\text{-}receive\text{-}premium\text{-}notification(S) \wedge memory(S) = null$
- $memory(read(S,1)) = 1$ if $memory(S) = null$

- $\text{discount}(\text{read}(S,1), P1, C1) = \text{true}$ if $\text{product-type}(S,C1) = \text{regular} \wedge \text{memory}(S) = 1$ and
- $\text{memory}(\text{read}(S,1)) = \text{null}$ if $\text{memory}(S) \neq \text{null}$

It is important to mention that in the case of multiple ECA rules, the semantics of events is non-deterministic, i.e. in an arbitrary state arbitrary events can be applied. On the other hand, actions are translated to deterministic behaviour.

3.3 Complex Events and OTSs

In this section we will attempt to define the semantics of a complex event algebra as an algebra for a (timed) OTS. We chose the event algebra of [5] for reference. We will define the semantics for some standard event algebra operators. In order to do this however we must first introduce the notion of an observer group in a (timed) OTS.

Definition 6. *Assuming a (timed) OTS S , we define that the transition $\tau \in T$ belongs to the Observer Group $og = \{o_1, \dots, o_n\} \subseteq O$ iff $\forall o \in O \setminus og, o(\tau(u, d_1, \dots, d_n), v_1, \dots, v_n) = o(u, v_1, \dots, v_n)$.*

Now assume that in our OTS the semantics of primitive events are those of transitions, meaning that each primitive event A , is mapped to a transition rule in the OTS. The proposed event algebra of [5] consists of the following 5 complex event operators; *disjunction, conjunction, negation, sequence* and *temporal restriction*. In the following definition we specify the semantics of these operators in the OTS framework inductively.

Definition 7. *Each transition rule that denotes a primitive event is a complex event. Assuming complex events A and B with effective conditions c_A and c_B respectively we define the following:*

The disjunction transition rule $A \vee B : S D_{A_1} \dots D_{A_n} D_{B_1} \dots D_{B_m} \rightarrow S$, with effective condition $c_A \vee c_B$. Where $\forall o \in O, o(A \vee B(u, d_{A_1}, \dots, d_{A_n}, d_{B_1}, \dots, d_{B_m}), v_{o_1}, \dots, v_{o_k}) = o(A(u, d_{A_1} \dots d_{A_n}), v_{o_1}, \dots, v_{o_k})$ if c_A or $o(A \vee B(u, d_{A_1}, \dots, d_{A_n}, d_{B_1}, \dots, d_{B_m}), v_{o_1}, \dots, v_{o_k}) = o(B(u, d_{B_1} \dots d_{B_n}), v_{o_1}, \dots, v_{o_k})$ if c_B . Meaning that either event A happens or event B , but not both.

The sequence transition $A; B : S D_{A_1} \dots D_{A_n} D_{B_1} \dots D_{B_m} \rightarrow S$ as the composition of transitions A and B . Such that $\forall o \in O o(A; B(u, d_{A_1} \dots d_{A_n}, d_{B_1} \dots d_{B_m}), v_{o_1}, \dots, v_{o_k}) = o(A(B(u, d_{A_1} \dots d_{A_n}), d_{B_1} \dots d_{B_m}), v_{o_1}, \dots, v_{o_k})$. With effective condition $c_{A;B} = c_B(S) \wedge c_A(S')$, where S' is the successor state of S when transition B is applied to it.

The conjunction transition rule $A+B$, denoting that both events occur. If the events occur simultaneously then for the system to be able to observe them they have to belong to different observer groups. So we define $A+B : S D_{A_1} \dots D_{A_n} D_{B_1} \dots D_{B_m} \rightarrow S$, such that $\forall o \in og_A; o(A+B(u, d_{A_1} \dots d_{A_n}, d_{B_1} \dots d_{B_m}), v_{o_1}, \dots, v_{o_k}) = o(A(u, d_{A_1} \dots d_{A_n}), v_{o_1}, \dots, v_{o_k})$ and $\forall o \in og_B; o(A+B(u, d_{A_1} \dots d_{A_n}, d_{B_1} \dots d_{B_m}), v_{o_1}, \dots, v_{o_k}) = o(B(u, d_{B_1} \dots d_{B_n}), v_{o_1}, \dots, v_{o_k})$. Where og_A and og_B are the observer groups of transitions A and B respectively. Note that $og_A \cap og_B = \emptyset$.

If the events do not occur simultaneously then $A+B$, $B+A$ are equivalent to the sequential complex events $A;B$ and $B;A$ respectively.

The negation transition $A-B$ that denotes the state where there is an occurrence of event A while event B does not occur. Occurrence of an event A denotes that the observers effected by transition A have the same values as when A is applied to an arbitrary state. In this case we wish to define a new transition rule stating that while all the observers in the observer group of A return the same values as those returned by applying A to an arbitrary state it is not possible for event B to occur. Since we are in a Timed OTS (TOTS) we have two associated observers, l_τ and u_τ for each transition τ denoting the lower and upper time bound of the transition rule respectively. So it suffices to define $A - B : SD_{A_1} \dots D_{A_n} \rightarrow S$ such that $\forall o \in O, o(A - B(u, d_{A_1} \dots d_{A_n}), v_{o_1}, \dots, v_{o_k}) = o(A(u, d_{A_1} \dots d_{A_n}), v_{o_1}, \dots, v_{o_k}) \wedge l_B(A - B(u, d_{A_1} \dots d_{A_n})) = \infty$.

Finally, we define temporal restrictions, i.e. an occurrence of an event A shorter than τ - time units. In the TOTS framework the effective condition of transitions tick basically forces the time to stop advancing if it will surpass the upper bound of any transition rule. So we define the complex event $A\text{-time} : SD_{A_1} \dots D_{A_n} \mathbb{R}^+ \rightarrow S$ such as $\forall o \in og_A, o(A\text{-time}(u, d_{A_1} \dots d_{A_n}, \tau), v_{o_1}, \dots, v_{o_k}) = o(A(u, d_{A_1} \dots d_{A_n}), v_{o_1}, \dots, v_{o_k})$ under the same effective condition as transition A . Also for all other primitive or complex events τ that belong to the same observer group as A , we define that $l_\tau(A\text{-time}(u, d_{A_1} \dots d_{A_n}, \tau)) = \text{now}(A\text{-time}(u, d_{A_1} \dots d_{A_n}, \tau)) \wedge u_\tau(A\text{-time}(u, d_{A_1} \dots d_{A_n}, \tau)) = \text{now}(A\text{-time}(u, d_{A_1} \dots d_{A_n}, \tau)) + \tau$. Through the second half of the last predicate we make sure that A will no longer occur after τ -time units, since the clock will be stopped until a transition of the same observer group as A is applied successfully.

3.4 Knowledge Representation (KR) Rules and OTSs

Knowledge representation focuses on the inferences that can be made from the fact that certain events are known to have occurred or are planned to happen in future. Among the KR formalisms, are the event calculus [6], the situation calculus [7], various action languages and event logics. In this paper we will focus on:

- Situation Calculus (SC)
- Event Calculus (EC)

In SC approach we assume a set of properties of interest for the system, say $\{P_1, \dots, P_n\}$. Now assuming an arbitrary system state S and an action of the system we define that if action A occurs in situation S a new situation results ($\text{result}(A, S)$). In ($\text{result}(A, S)$) property $P \in \{P_1, \dots, P_n\}$ will be *true (false)* if action A in state S *initiates (terminates)* P .

This can be defined as follows :

- $P : \text{true} \rightarrow A \text{ initiates } P \text{ in } S$
- $P : \text{false} \rightarrow A \text{ terminates } P \text{ in } S$

Here, with SC we mean the original version of McCarthy [7] and not of R. Reiter, i.e. a situation is a state or a snapshot rather than a sequence of actions.

Definition 8. *The formalization of such a system in the OTS approach can be done using the OTS $S = \langle O, I, T \rangle$ where:*

- $T = \cup\{A_i\}$ a finite set of transitions that correspond to the finite set of actions defined by the rules. Each such transition is defined as $A_i : Sys D_1 \dots D_{k_i} \rightarrow Sys$
- I a set of initial states
- O is the set of observers $\{initiated, terminated\} \cup \{P_i\}$. Where $initiated : Sys Label1 Label2 \rightarrow Bool$, $terminated : Sys Label1 Label2 \rightarrow Bool$ and $P_i \in \{P_1, \dots, P_n\}$ with $P_i : Sys D_{1_i} \dots D_{n_i} \rightarrow Bool$. In the previous we assume the predefined visible sorts $Label1$ and $Label2$ that denote the actions of the system and the properties of interest respectively. The first observer returns true when action A initiates property P and the second observer returns true when action A terminates property P . This is formalized by the following equations; $P_j(A_k(S, v_{1_i}, \dots, v_{n_i})) = true$ if $c-A_k(S) \wedge initiated(S, p_j, a)$. Also $P_j(A_k(S, v_{1_i}, \dots, v_{n_i})) = false$ if $c-A_k(S) \wedge terminated(S, p_j, a)$. Where, the constants a, p_j denote an arbitrary action and a property j respectively, while $v_{1_i} \dots v_{n_i}$ denote arbitrary visible sorts values needed for the definition of the transitions.

In EC a model of change is defined in which events happen at time points and initiate and/or terminate time intervals over which certain properties of the world hold. The basic idea is to state that properties are true at particular time points if they have been initiated by an event at some earlier time point and not terminated by another in the meantime.

Definition 9. *This notion can be formalized in the OTS approach as well using a TOTS $S = \langle O, I, T \cup \{tick_r\} \rangle$ defined as follows.*

- T is the set of transitions such that $T = \cup\{A_i\}$ (a finite set of transitions that correspond to the finite set of actions defined by the rules). Each such transition is defined as $A_i : Sys \rightarrow Sys$ and $\{tick_r\}$ the usual tick operators defined in the generic TOTS.
- I is a set of initial states
- O the set of observers, where $O = \{initiated, terminated, now\} \cup \{P_i\}$, with $initiated, terminated$ and P_i as in definition 8. Also "now" is a special observer whose signature is $now : S \rightarrow \mathbb{R}^+$ and denotes the systems master clock. The equations defining the observers in an arbitrary state are (adopting the same notation as definition 8): $P_j(A_k(S, v_{1_i}, \dots, v_{n_i})) = true$ if $\exists S', A_m$ such that $initiated(S', p_j, a_m) = true \wedge now(S') \leq now(S) \vee \nexists S', A_m$ such that $terminated(S', p_j, a_m) = true \wedge now(S') \leq now(S)$. For the symmetrical observer ($terminated$) we have the equations:
 $P_j(A_k(S)) = false$ if $\nexists S'', A_m$ such that $(initiated(S'', p_j, a_m) = true \wedge now(S'') \leq now(S)) \vee \exists S'', A_m$ such that $(terminated(S'', p_j, a_m) = true \wedge now(S'') \leq now(S))$

In languages implementing OTSs quantifiers need to be treated carefully. This is due to the fact that these languages usually rely on equational logic. There, the \forall quantifier can be handled by free variables, i.e. each equation $E(x)$ containing an unbound variable x is semantically equivalent to $\forall xE(x)$. On the other hand the \exists quantifier is not straightforwardly supported. However, each equation containing an \exists quantifier can be transformed into its equivalent Skolem normal form without such quantifiers.

Definition 10. *The equations defining the observers in the above definition, can be replaced with the following for a language that implements an OTS.*

- $P_j(A_k(S, v_{1_i}, \dots, v_{n_i})) = \text{true}$ if $\text{initiated}(f_{S'}(S), p_j, f_{a_m}(S)) = \text{true} \wedge \text{now}(f_{S'}(S)) \leq \text{now}(S) \vee \text{terminated}(f_{S'}(S), p_j, f_{a_m}(S)) = \text{true} \wedge \text{now}(f_{S'}(S)) \leq \text{now}(S)$.
- $P_j(A_k(S)) = \text{false}$ if $(\text{initiated}(f_{S'}(S), p_j, f_{a_m}(S)) = \text{true} \wedge \text{now}(f_{S'}(S)) \leq \text{now}(S)) \vee (\text{terminated}(f_{S'}(S), p_j, f_{a_m}(S)) = \text{true} \wedge \text{now}(f_{S'}(S)) \leq \text{now}(S))$.

Where $f_{S'}$ and f_{a_m} are the Skolemization functions that map each hidden sort to a hidden sort and each hidden sort to a label 2, respectively.

The formalisation for EC presented here corresponds to Simplified Event Calculus (SEC), i.e. we employ time points instead of time periods. A similar approach however could be adopted for the original EC as well.

4 Conclusions and Future Work

We presented an OTS semantics for production (PR), event condition action (ECA) and knowledge representation rules (KR) as well as for complex event processing. Our goal is to have a unifying theory and appropriate tool support for it, built using algebraic specification languages (like CafeOBJ [11] or Maude [13]). In the future we expect to extend this work by mapping Reaction RuleML to one of the above languages and develop tools that automate this process. Verification support for Reaction RuleML rule bases will lead to the creation of online libraries and the definition of operators for combining these rules in a way that preserves the desired properties.

Also the formal proofs of the verified properties are executable [12] and could be used as trust credentials for the rule bases agents use, thus allowing the creation of new trust policies and models. Finally, verification techniques could be applied to rule engines and prove the correctness of the implementation of the rules.

Acknowledgments. The authors would like to thank Prof. Harold Boley for his valuable ideas and comments. Without him, this work would not have been possible.



This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) – Research Funding Program: THALIS

References

1. Paschke, A., Boley, H.: Rules Capturing Events and Reactivity. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 215–252. IGI Publishing (May 2009)
2. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: The Overarching Specification of Web Rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010)
3. Goguen, J.A., Diaconescu, R.: Towards an Algebraic Semantics for the Object Paradigm. In: Ehrig, H., Orejas, F. (eds.) 10th Workshop on Abstract Data Types (1994)
4. Ogata, K., Futatsugi, K.: Proof Scores in the OTS/CafeOBJ Method. In: Najm, E., Nestmann, U., Stevens, P. (eds.) FMOODS 2003. LNCS, vol. 2884, pp. 170–184. Springer, Heidelberg (2003)
5. Carlson, J., Lisper, B.: An event detection algebra for reactive systems. In: 4th ACM International Conference on Embedded Software (2004)
6. Kowalski, R.A., Sergot, M.J.: A logic-based calculus of events. *J. New Generation Computing.* 4, 67–95 (1986)
7. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Michie, D., Meltzer, B. (eds.) Machine Intelligence 4, pp. 463–502. Edinburg University Press (1969)
8. Goguen, J.A.: Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 217–273 (1992)
9. Ogata, K., Futatsugi, K.: Modeling and verification of real-time systems based on equations. *Science of Computer Programming* 66, 162–180 (2007)
10. Paschke, A., Kozlenkov, A., Boley, H.: A Homogeneous Reaction Rule Language for Complex Event Processing. In: VLDB 2007 (2007)
11. CafeOBJ Homepage, <http://www.theta.ro/cafeobj/>
12. Futatsugi, K., Babu, C. S., Ogata, K.: Verifying Design with Proof Scores. In: Meyer, B., Woodcock, J. (eds.) VSTTE 2005. LNCS, vol. 4171, pp. 277–290. Springer, Heidelberg (2008)
13. Maude Homepage, <http://maude.cs.uiuc.edu/>
14. Diaconescu, R., Goguen, J., Stefaneas, P.: Logical support for modularization. In: Second Annual Workshop on Logical Environments (1993)
15. Goguen, J., Malcolm, G.: A hidden agenda. *Theoretical Computer Science* (1996)

A Rule-Based Calculus and Processing of Complex Events

Stefano Bragaglia, Federico Chesani, Paola Mello, and Davide Sottara

DEIS, University of Bologna,
Viale Risorgimento n. 2
40136 - Bologna, Italy
{Stefano.Bragaglia,Federico.Chesani,
Paola.Mello,Davide.Sottara}@unibo.it

Abstract. Rules are definitely among the main kinds of knowledge representation in Artificial Intelligence. In recent years, there has been much discussion about production rules and logic programming to understand whether the two paradigms could be joined or, alternatively, which was the better. Conversely, the idea to program a production system with logic without actually relying on logic programming was proposed.

In this paper we present a software component that implements a typical logic formalism, the *Event Calculus* within a production rules system. This component allows to perform deductive reasoning tasks (temporal projection or prediction, such as monitoring) and thanks to some technical choices, it proves to be quite efficient. In addition, thanks to its strong modular nature, it can adapt to the domain's requirements and complement other forms of reasoning at the same time.

We also present some preliminary results on tests that we have conducted to show that our system based on a Java rules engine is almost as efficient as an equivalent logic program running on the fastest C++ Prolog interpreter. Furthermore we show how our framework can be used to effectively observe the evolving state of our use case – a *Service Oriented Architecture* server – in a way that qualifies as *Complex Event Processing*.

Keywords: Complex Event Processing, Event Calculus, Production Rules, Rule-based Reasoning.

1 Introduction and Motivations

“Rules” are considered the basic form for representing the knowledge in many areas of Artificial Intelligence. Among the possible different types of rules, the production rules and the logic programs are probably the most common. Despite both of them are widely used, there is a great deal of confusion and disagreement about the different kinds of rules and their mutual relationship [1].

The production systems' rules have the form *if conditions then actions* and appears to be similar to conditionals in logic. The most popular textbook on Artificial Intelligence [2, p. 286], in fact, considers production rules as mere

conditionals for forward reasoning. One of the main textbooks on Cognitive Science [3, p. 43], however, asserts that “rules are if-then structures” that, despite being “very similar to the conditionals”, “they have different representational and computational properties”. With respect to Prolog, it is presented as “a programming language that uses logic representations and deductive techniques” [4], but it is also included “among the production systems widely used in cognitive simulations” [3].

Logic systems like Prolog, in fact, are *backward-chaining* tools: starting from the goal, they apply logic clauses by inferring heads when bodies unify with current goal until facts are reached. Such process, known as *Selective Linear Definite clause resolution with Negation as Failure (SLDNF)*, is performed by investigating a “derivation” for the goal at a time, possibly backtracking to try others when it does not lead to any solution. Production rule systems, instead, are typically *forward-chaining* tools: by considering the facts that are currently known, they apply the consequences of those rules whose premises match the facts. Such consequences may result in new knowledge that could trigger the rules again. Thus, “threads of reasoning” here are built in the reversed order with respect to “derivations”, but they are produced all together in parallel. With respect to the case of the *Event Calculus (EC)*, for example, the latter aspect is particularly successful when the status of many fluents is queried at the same time as it typically happens with *Complex Events Processing (CEP)*. As a drawback, those systems are usually quite complex and some optimisations may make their semantics a bit loose¹.

With these premises, it is easy to see why these two worlds were kept separate for a long time. Recently, however, the idea of using systems that are not strictly logical to perform logic reasoning has been proposed. The logical systems have the advantage of being able to prove formal properties of the reasoning (a goal that is still possible with non-logical tools like production rules systems if they are programmed in a proper “logical” manner [5]) but often they lack other properties. According to such perspective, in this paper we present our implementation of the EC – a framework that has been typically associated to logic programming – embedded in a system to assist the processing of complex events by means of production rules. Such a result is not trivial because the EC machinery must be fast enough in order to blend in with the CEP philosophy. Moreover, the machinery must also be self-contained and independent not to clash with any other reasoning that is going on the same shared domain data.

The remainder of this work is organised as follows: in Section 2 we introduce the EC and some of its variants, in Section 3 we present some desiderata for our rule-based implementation, with a deep discussion on its architectural outline and the results of some performance tests, Section 4 describes our use case – a *Service Oriented Architecture* server – and shows how we have tackled it and, finally, Section 5 draws some conclusions and outlines the future works.

¹ Consider for example Conflict Resolution – the process that decides which activated rules are executed first – for which it is difficult to demonstrate its non-determinism.

Table 1. The Event Calculus ontology

Axiom	Meaning
$holdsAt(F, T)$	Fluent F holds at time T
$initially(F)$	Fluent F holds from the initial time
$happens(E, T)$	Event E happens at time T
$initiates(E, F, T)$	Event E initiates fluent F at time T
$terminates(E, F, T)$	Event E terminates fluent F at time T
$clipped(F, T_1, T_2)$	Fluent F is terminated by an event in (T_1, T_2)

2 The Event Calculus

The *Event Calculus* (*EC*) is a logic-based formalism for representing and reasoning about actions and their effects. It was introduced by Kowalski and Sergot in 1986 [6] and later extended by Shanahan and Miller in the 90's [7]. As for other similar languages (the most prominent of which is probably the *Situation Calculus* (*SC*) [8]), its basic elements are the *events* and the *fluents*. An event is any thing that occurs on a domain at any given time, that causes at least a partial change of its state. A fluent instead is any measurable aspect of the domain that is subject to changes over time (the set of the values of a domain's fluents is therefore its state). Together they are used to describe what is happening over time, their effects and, ultimately, how a domain evolves.

The essence itself of the EC is contained in the following sentence that describes its main operating principle: a fluent is *true* in a given time instant t *iff* it was initially true or it has been made true in the past and has not been made false in the meantime. Most implementations of the EC were defined in terms of the Horn subset of classical logic augmented with negation as failure to make them straightforwardly executable as Prolog programs. Thus, the above principle translates into the following clauses:

$$\begin{aligned}
 holdsAt(F, T) \leftarrow \\
 \quad initially(F), T_0 < T, \neg clipped(F, T_0, T). \tag{1}
 \end{aligned}$$

$$\begin{aligned}
 holdsAt(F, T) \leftarrow \\
 \quad happens(E_i, T_i), initiates(E_i, F), T_i < T, \neg clipped(F, T_i, T). \tag{2}
 \end{aligned}$$

$$\begin{aligned}
 clipped(F, T_s, T_f) \leftarrow \\
 \quad \exists E, T : [happens(E, T), terminates(E, F), T_s < T, T < T_f]. \tag{3}
 \end{aligned}$$

Notice that this core axiom needs to be supplemented by some auxiliary domain-dependent axioms to provide enough information to complete the domain model. The Table 1 contains the list of all the predicates that the user may safely use to model a domain and interpret the results without tampering with the calculus itself and their meaning. The Fig. 1, instead, suggests their purpose: *initiates*/3 and *terminates*/3 (*what events do*) express behavioural information, *initially*/1 and *happens*/2 (*what happens when*) which together take the name of *narrative*

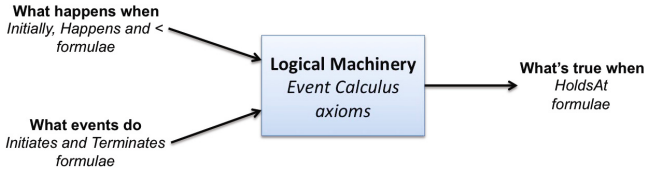


Fig. 1. Operating diagram of the Event Calculus for deductive tasks

or *trace* define temporal information and finally *holdsAt/2* and *clipped/3* (*what's true when*) for the core axiom. Notice that an additional $<$ axiom is required to determine temporal precedence: its definition, however, is not directly relevant, provided that it satisfies some integrity constraints such as *transitivity* and *anti-symmetry* [9]. Consider the following example to see how the EC can control the evolution of a domain.

Example. The light switch scenery is a classic example in EC research literature. A fluent *switchOn* reflects the status of a switch anytime it is toggled by a *turn off* or a *turn on*. Similarly, a fluent *pwrAvail* reveals whether power is available after a *power failure* or a *power restore*. Finally, a fluent *lightOn* becomes true anytime *switchOn* and *pwrAvail* are contextually true (or false in any other case). If the power is initially available and the switch is turned off, each fluent's status may be determined when new events happen:

initially(pwrAvail).

initiates(turnOn, switchOn, T) ← happens(turnOn, T).

terminates(turnOff, switchOn, T) ← happens(turnOff, T).

initiates(pwrRest, pwrAvail, T) ← happens(pwrRest, T).

terminates(pwrFail, pwrAvail, T) ← happens(pwrFail, T).

initiates(turnOn, lightOn, T) ← happens(turnOn, T), holdsAt(pwrAvail, T).

terminates(turnOff, lightOn, T) ← happens(turnOff, T).

initiates(pwrRest, lightOn, T) ← happens(pwrRest, T), holdsAt(switchOn, T).

terminates(pwrFail, lightOn, T) ← happens(pwrFail, T).

The EC has been also defined as “a logical mechanism that infers **what's true when given what happens when and what actions do**” [10]² (see Fig. 1). It is not difficult to see, however, that it can provide other styles of reasoning just by reversing the direction of some arrows. We have **deductive reasoning**, for example, if we use “*what happens when*” and “*what events do*” to determine “*what's true when*”; it includes temporal *projection* or *prediction* to determine the outcome of a known sequence of actions. The **abductive reasoning**, instead, exploits “*what events do*” and “*what's true when*” to find out “*what happens when*”; referring to temporal *explanation* or *postdiction*, certain kinds of *diagnosis*

² The Author explicitly states that he uses the terms *action* and *event* interchangeably.

and *planning*, it is used to derive sequences of actions that lead to a desired state. Finally, the *inductive reasoning* focuses on “*what’s true when*” and “*what happens when*” to return “*what events do*”; it can perform certain kind of *learning*, *scientific discovery* and *theory formation* and, generally, it supplies a set of rules or a theory on the effects of actions that accounts for observed data. The tool presented here, of course, can only perform deductive reasoning, however by introducing the concept of *desired future state* and by properly expanding the ruleset, it may support abductive reasoning as well. By opportunely partitioning the knowledge base, in fact, the two reasoning styles may coexist. Such approach has been already discussed in [5] and it will be object of future work.

According to [10,9,11], the most common and basic variant of the EC is probably the *Simple EC (SEC)* [12], so called because it is based on time instants rather than time intervals (as instead the original formulation was [6]) and therefore is less complicated. Numerous extensions to SEC have been proposed in order to provide additional features such as, for example, the ability of properly handling concurrent events and continuous changes that were introduced by the *Extended EC (EEC)* (sometimes also indicated as *Basic EC* by some authors) [10,13]. This variant, for example, is particularly appealing for our use case because it provides predicates to handle conflicting events that may happen in such a crowded domain with poor temporal granularity. These extensions, however, suffer from the same drawback: anytime the notification of a domain’s event is received, the EC computes the new result from scratch.

Instead, according to the *common-sense law of inertia* [14], small updates on input (like the happening of an event) should cause marginal alterations on output. This suggests that an incremental algorithm for the EC will ensure better performance. Such a limitation has been pointed out for the first time and addressed by Chittaro and Montanari with their proposal of a *Cached EC (CEC)* [15]. As the name suggests, all the fluent’s history is tucked away so that all the subsequent updates are simply added on top of it. Later, a simplified and more efficient version of the CEC called *Reactive EC (REC)* was proposed [16]. This variant is ideal for all those domains where the notifications of the events’ occurrences are received in proper chronological order. Both CEC and REC introduce a new predicate $mvi(F, [T_1, T_2])$ (not included in Table 1 and in *what’s true when* side of Fig. 1) which models the *maximal validity intervals* during which fluents uninterruptedly hold. Notice however that, despite being generally faster than any standard EC variant, their implementations relies on the *assert/1* and *retract/1* predicates which have no underlying declarative semantics in Prolog and cannot prove formal properties of the calculus. This is not necessarily a problem in almost any context, but it may be a deal breaker in a few cases.

For all these reasons, the EC is still one of the most used formalisms to reason about the effects on a domain over time of the events, even after 25 years from its introduction. It has been exploited, for example, in a variety of domains, such as cognitive robotics [17], planning [18], service interaction [19] and composition [20], active databases [21], workflow modelling [22] and legal reasoning [23].

Moreover, in addition to the original formulation [6,12], the EC has been proposed in many other variants and extensions, based on both logics [9,10,13,11,15,16] and other programming paradigms [23,15]. Nowadays networks, computers and communications are so large and pervasive that their complexity is typically tackled by decomposing them in smaller, possibly distributed entities. With respect to deductive reasoning – the reasoning style that will be addressed in the remainder of this paper, the distributed software approach is very suitable to be modelled in terms of EC. Examples of this approach are Business Process Management [24], (Computerised) Clinical Guidelines [25], Service-Oriented Computing [26] and Multi-Agent Systems [27].

3 A Rule-Based Incremental Event Calculus

In order to check the evolving state of a running system and, ultimately, to perform real-time monitoring on it, it is appropriate to exploit a deductive reasoning framework. As we have suggested in Section 2, the EC is one of the formalisms that is widely considered to be among the best choices to which to resort for this kind of tasks. As seen, most existing implementations rely on Prolog and usually take only advantage of the more basic variants of the EC. The adoption of Prolog guarantees the provability of some formal property of the calculus but it is typically detrimental for performances. This is not necessarily due the adopted technological platform but rather on the way these tools were implemented on top of that platform. They were probably built with the idea to prove the feasibility of the calculus rather than to provide an efficient solution. Aspects like the adoption of a basic variant or the retention of all the objects in memory become a bottleneck for the reasoning especially for larger complex domains, and they will be discussed with more details in Section 3.2.

Conversely, our implementation is based on *Drools*³ which is an efficient well-known *forward chaining* rules engine based on the *RETE* algorithm [28]. More properly, it is an open source suite for the integration of *knowledge modelling* and *business logic* composed by several modules. One of them, *Drools Fusion*, is responsible for *complex event processing (CEP)* [29] and reasoning over *temporal intervals* [30]. This rule engine has a blackboard-like memory called *Working Memory (WM)* where *facts* (simple Java objects) about the domain are handled. Some objects may even be declared as Fusion's *events*, meaning that they are automatically decorated with temporal information that is used to reason about time. The engine also has a *Production Memory (PM)* which contains the conditional statements known as *rules* that implement the processes that govern the domain. The rules are composed by a *pattern* and a set of *actions*: when some facts match with a rule's pattern, that rule is *activated* by those facts and its actions are subsequently executed. The patterns may be composed of several facts whose values may of course be compared using *operators* (boolean, relational, equivalence, temporal, etc.). The rules can obviously be fired in cascade to

³<http://www.jboss.org/drools>

```

declare Maintenance
  @role( event )      @timestamp( start )      @duration( extent )
  routine : boolean   start : long             extent : long
end

rule "CEP Rule Example"
when
  Number( $value : longValue > 2h ) from accumulate (
    Maintenance( routine == false, $extent : extent ) over window:time( 7d ),
    sum( $extent )
  )
then
  log("Weekly limit for maintenance exceeded: " + $value);
end

```

Fig. 2. A simple Drools Fusion theory detecting excessive weekly maintenance

shape deeper and more sophisticated reasonings. The temporal reasoning is refereed by a *clock* that maintains the notion of time and determines which events pertain to a given time interval. This clock can be instantiated as a *pseudo* or *real-time* clock depending on whether you want to control the flowing of time during simulations or to build online reactive systems. The following example aims to provide an insight of how to model domains in Drool and its potential.

Example. Consider the code in Fig. 2 which shows how a contractor might detect violations on an agreement regulating unexpected maintenance. Each intervention is represented by an instance of **Maintenance**: a Fusion’s event whose times are automatically set when the object enters the WM and on job done (by another agent who sets the **extent** according to the some **Report** that has not been modelled into the example for sake of simplicity). The **routine** attribute distinguishes habitual works from unexpected ones. The PM contains only a rule which searches for unexpected interventions that occurred in the last weekly time interval and accumulates their duration. If the final figure exceeds the two hours threshold, the agreement is then infringed and a notification is flagged.

3.1 Architectural Outline of the Tool

Before starting to work on the implementation of our software component for the deductive reasoning, we have identified some desiderata that we set as non-functional requirements in order to obtain a more usable, robust and powerful system. In particular, we were aiming to a module as *efficient*, *generic but easily customisable* and *independent* as possible. Notice that, in regards of the latter requirement, we expressly aimed to be complementary with other extensions that we have already proposed such as the fuzzy semantic one [31] and the one supporting expectations [32].

We have decided then to organise the concepts of EC into several groups: some of them where freely available to the user to define domains, while others were not. The idea to use this sort of layering – which is known as *stratification* – was borrowed from other implementations where the most sensible data for the calculus were hidden from the user not to let him directly interfere with

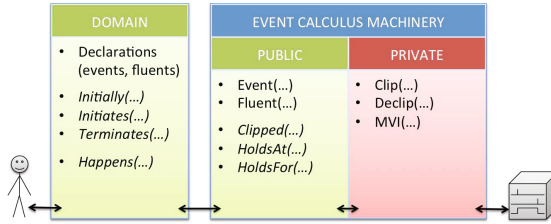


Fig. 3. Stratification of EC axioms for a proper use of the machinery

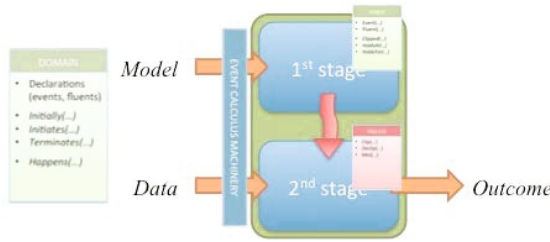


Fig. 4. Architectural pattern adopted for the tool which shows what subsystem handles the knowledge pertaining to each layer of the stratification

the reasoning mechanism itself. The terms pertaining to the red area in Fig. 3, for example, are labelled as “private” because only the engine can use them to represent the domain’s status. The green area, instead, contains all the terms that can be safely used: this area has been split into two parts because some concepts may be totally used as the user pleases (*read and write access*), while other may be just referred (*read-only access*). In particular, the former are labelled as “domain” and the latter – that together with the private ones provide the core definitions for operating the EC machinery – as “public”. The terms in the green area that are not in italics have been explicitly added to introduce **Event** (implemented as a Drools Fusion’s event) and **Fluent** (a simple object holding a Boolean **status** field and a reference to the last **Event** which had affected it) as top-level concepts: users just have to extend to specify their domains. Notice also how all these terms recall the concepts listed into Table 1 (plus *mvi/2*, as seen in Section 2, and *holdsFor/3* added for user’s convenience to refer to a fluent holding uninterruptedly over a given time interval): in this case, however, they are all WM’s facts rather than logical predicates.

From a structural viewpoint, our Java tool is organised according the architectural pattern presented in Fig. 4. This pattern, which we have already exploited a few times in similar works, consists of two cascading stages. The former one is used to process part of the domain knowledge: the result of this preprocessing is passed to the 2^{nd} stage that actually provide the expected reasoning by coupling


```

rule "Transforming rule for 'Initiates' objects"
when
  Initiates(
    $e: event, $ep: eventPtrn, $f: fluent, $fp: fluentPtrn, $c: context )
then
  Helper h = Helper.getInstance(); // a wrapper to ease conversion
  String r = h.transform($e, $ep, $f, $fp, $c);
  h.insertRule(r);
end

rule "Resulting rule"
when
  $e: TurnOn( $t: time )
  $f: lightOn() // a query returning this fluent instance
  holdsAt( powerAvail(), $t )
then
  insert(new Declip($e, $f, $t));
end

```

Fig. 5. The transformation rule that converts `Initiates` objects into rules that trigger the EC core upon events' occurrence

the transformed data with the remaining domain knowledge. Notice that some complex domains may require additional cascading stages, moreover logically distinct stages may practically coexist in the same WM⁴.

Fig. 4 also shows which component handles the distinct pieces of information identified with stratification. The 1st stage provides declarations of the base **Event** and **Fluent**: the user extends them to declare domain's events and fluents; such declarations are passed to the 2nd stage where the calculus is actually done. The user has also to insert a few **Initially**, **Initiates** and **Terminates** objects into the 1st stage's WM to fully define the model⁵. The former kind of objects tells the system which **Fluent** among those previously introduced initially holds: this is trivially addressed by a transformation rule that inserts a **Declip** object (see later) with `time` equals to 0 into the 2nd stage's WM for the given **Fluent**.

The latter objects define the causal relationship between events and fluents' values: each **Initiates** (**Terminates**), for example, tells the system that when an **Event** of the given type happens at time t , then a **Declip** (**Clip**) object for the given **Fluent** with the same timestamp t must be inserted into the 2nd stage's WM, if the given `context` is satisfied. The `context` is usually a combination of constraints that may imply, for example, a delay between the event and the commutations of target fluent's state or a given compound status for a few some other fluents at a certain time. A couple of transformation rules converts these objects into rules that trigger on instances of domain's **Events**. These events are directly fed into the 2nd stage by some external "sensor" agents. Last but not least, the 2nd stage also contains the declarations and rules

⁴ Nevertheless, the data can be kept separate in Drools by partitioning the WM with *entry points*: this good practice allows not to mess with data when several reasoning tasks are being performed at the same time.

⁵ It has been proven, in fact, that the EC may return incorrect results whether the knowledge on the domain is incomplete [10].

```

// Lite mode core rules
query holds( Fluent $f )
  $f := Fluent( status == true )
end

rule "Clipping an Event"
when
  $c: Clip( $e: event, $f: fluent )
  $f := Fluent( status == true )
  not Declip( fluent == $f,
             $c before this )
then
  modify($f) {
    setEvent($e),
    setStatus(false);
  }
end

// Full mode core rules
query holdsAt( Fluent $f, long $t )
  MVI( fluent == $f, start < $t,
       stop >= $t )
end

rule "Clipping a MVI"
when
  $c: Clip( $f: fluent,
           $s: start )
  $i: MVI( fluent == $f,
          start < $s, stop > $s )
then
  modify($i) {
    setStop($s);
  }
end

rule "Declipping a MVI"
when
  $d: Declip( $f: fluent,
             $s: start )
  not MVI( fluent == $f,
          start >= $s // stop >= $s )
then
  insert( new MVI( $f, $s,
                  Long.MAX_VALUE ) );
end
//-----

```

Fig. 6. The core rules for *Lite* and *Full* modes (complementary rules are omitted)

(discussed later) that make the EC machinery itself. The transformation rule for `Initiates` objects and the resulting rule for the object `Initiates(TurnOn, LightOn, holdsAt(PowerAvail, t))`⁶ is presented as an example in Fig. 5.

Notice that declared `Events` and `Fluents` may contain fields to distinguish between instances of the same kind across the domain (consider for example a circuit with a single generator but multiple switches and lights). This approach allows a more streamlined definition of the model but, as well as the `context`, it would require a proper language and parser to cope with it. Since this is not the goal of this work, we have decided to let the user express this information by means of strings whose content must be well formed Drools statements, patterns or queries. This sort of *loose-typing technique*, however, poses a consistency threat since the user must be consistent in all definitions.

The core of the module consists of a set of rules that works on the aforementioned `Clip` or `Declip` facts to update a domain's state. We have implemented two modes of operations for it that more or less resemble the two incremental EC variants seen in Section 2: REC and CEC. The user may choose the one he prefers at the beginning of each working session, according to his needs. The mode of operation that was inspired by REC has been named *Lite* because it requires only a few computational resources and it is very efficient when it is reasonable to assume that the events are notified to the system in the proper temporal order. In addition, this mode of operation purges any superfluous instance from memory upon applying its effects on the current state of the domain. This implies that it has a really small memory footprint allowing the computation of longer traces, but it may be used only to reason about the current state

⁶ t implicitly refers to the timestamp of the `TurnOn` instance that will trigger the resulting rule.

of the domain. The other mode of operation, instead, is similar to CEC since it exploits MVI objects to memorise the evolution over time of each `Fluent`'s `status`. We called it *Full* mode because it does not flush any information from the WM, it updates properly MVIs (even in case of delayed events) and it can report the `status` of any `Fluent` in any point of domains' history. Notice however that the size of the largest problems that can be handled is limited by the amount of available memory, as for many other implementations. The core rules for both the *Lite* and *Full* modes of operations are reported in Fig. 6.

3.2 Experimental Evidence

In order to assess the efficiency of our solution, we have conducted some preliminary tests to identify the fastest Prolog counterpart to be used later as a term of comparison. We chose the Example introduced in Section 2 as a test case and we prepared a trace that contains as much as 600 events. This trace has been used to build three narratives: in the first one, the events were presented in chronological order, in the second one in reverse chronological order and in the third one were scrambled as if some events were notified with some delay⁷. Then we conducted a quick survey on open source or freely available Prolog tools to assess the system on which to run the CEC and REC experiments. We have identified the following suites: *B-Prolog*, *SWI-Prolog*, *tuProlog* and *YAP Prolog*⁸.

On the ordered trace, REC clearly outperformed CEC since it was meant to do so by design. In the opposite scenario, CEC was performing better than REC but not so outstandingly as we had assumed before starting the tests. The third test being the most realistic and balanced, we decided to discard the other two and rely solely on its outcomes to carry on our further analysis. REC was generally quite faster than CEC with all the interpreters, with B-Prolog and YAP Prolog being the fastest. B-Prolog was actually even faster than YAP Prolog, but due to its aggressive strategy on memory allocation, some experiments were not able to complete for lack of memory. Thus we concluded to stick with YAP Prolog to perform the comparison with both the modes present in our system.

The execution times required to play events singularly and cumulatively are respectively presented in Fig. 7a and Fig. 7b. The Full and Lite mode appear to have trends similar to CEC and REC, with the latter two being only slightly faster. The Lite mode and REC are performing best and the Fig. 7b suggests that they have similar performance in the long run, with the Lite mode possibly becoming faster in larger problems. It appears, instead, that CEC is better on small problems (with less than 300 events in case of domains with a similar complexity as this one) but then the Full mode outperforms the other.

⁷ The delay was modelled by assuming that each event had the same probability of being received in time or after from 1 to 5 following events and by shifting events accordingly.

⁸ Respectively: <http://www.probp.com/>, <http://www.swi-prolog.org/>, <http://tuprolog.alice.unibo.it> and <http://www.dcc.fc.up.pt/~vsc/Yap/>

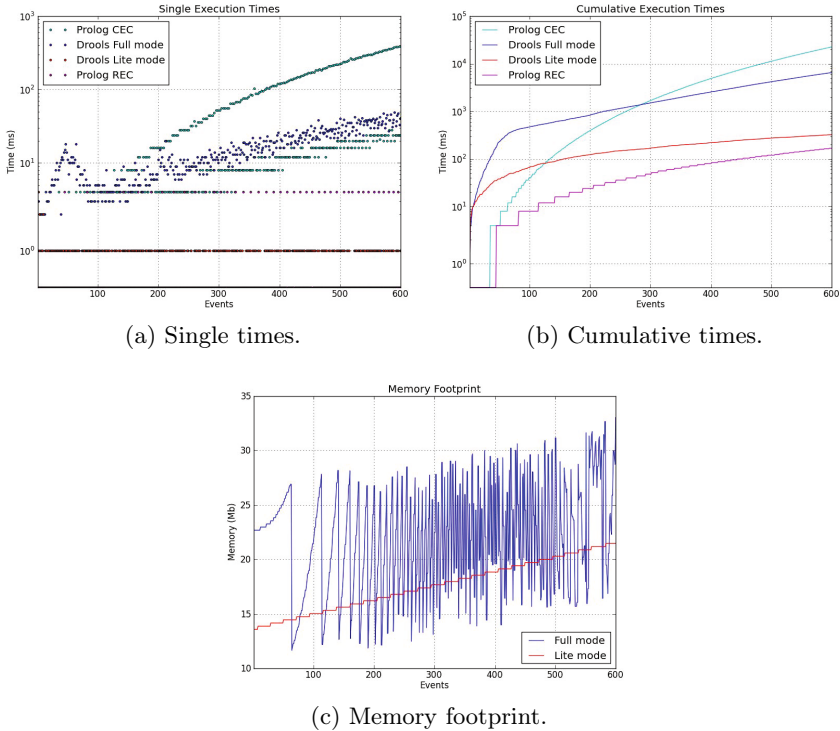


Fig. 7. Comparison benchmarks between *CEC*, *REC*, *Full* and *Lite mode*, carried out with *YAP Prolog 6.2* and *Drools 5.3* on an Intel i5 @ 2,4 GHz with 4GB

With respect to memory usage of the two modes, there are mainly two problems. Notice that this should not be regarded as a criticism toward logical rules in favour of production rule since both issues actually depends on implementation choices rather than the underlying technology. The first issue is of course the adoption of EC variants that are not incremental that lead to frequent freeing and allocation of memory every time filled with almost the same data. The other one is the retention strategy of objects in memory: many implementations in fact were aimed to demonstrate the feasibility and correctness of the calculus and later were used in contexts where, instead, other features were advisable. By reducing the number of objects in a WM and by marking them as “garbage collectible” soon after being processed, the memory footprint is sensibly reduced and larger domains can be handled. Fig. 7c compares the trend in memory usage of the two modes only: given the high amount of objects involved and its complete retentions strategy, Full modes shows a rough and hectic behaviour while Lite mode shows a nicer, more predictable pattern. Please notice that these results does not suggest that the Lite mode is always preferable to Full because, as we explained in Section 3.1, they are meant to address different needs.

To be honest, many real cases would require a third mode that is a compromise between the other two: a Full mode with memory limited to the recent past (be it a time window or a fixed cache of events). We managed to tackle this problem by configuring the system in Full mode and by manually including one or more rules (similar to the one in the example of Section 3) that clean the memory from the objects that are no more needed or too old. In a near future we could even promote such strategy to a new stand-alone mode.

4 Use Case: Observing a SOA Server

As a testbed suitable for both EC and CEP reasoning together, we focused on the *Service-Oriented Architectures (SOA)* domain by observing the state of a Web server. Instead of using a traditional software stack, we adopted a new technology called *Jolie* that we think is mature enough and very promising.

Jolie⁹ is a full-fledged programming language and development platform based upon the service-oriented programming paradigm, suitable to both the rapid prototyping of new services or the composition of existing ones to deliver new functionalities. It offers an easy to learn syntax, a formal theoretical semantics and a strongly modular approach. Thanks to its extensible development API, Jolie is suitable to make lightweight services, very complex SOA or bridge systems based on different technologies or communication standards. Such a highly customisable is prone to be monitored to verify it is conformant to user's requirements and to adapt its future operations according to its past performances. Despite this is definitely matter of future work, now we simply want to asses that we can observe such a complex domain and process all its data.

The Jolie interpreter relies on a virtual machine which is similar to Java's and capable of both deploying new services on the fly and keeping track of each working session and all their inner operations. We have modelled our EC knowledge base accordingly: we extract the notification of initiation and termination of each session and all the operations in it and we feed it to the deductive reasoner as events; sessions and operations become fluents whose activity intervals have to be determined. At the moment, additional information such as the amount of free memory is extracted but it is not involved in any other computation. The information processed in this way by the module EC is then forwarded to a separated Jolie service that we have implemented to display the outcome of the calculus. Fig. 8 contains a small excerpt of an execution trace processed by the EC reasoner as displayed by this service. Finally we have prepared an additional script service that orchestrates calls to the services deployed on the Jolie VM in a repetitive pattern. We had evidence that the deductive EC implementation managed to handle all the information originated by the Jolie interpreter during a real(-istic) working session. We have to admit, however, that the services deployed on the testing machine were trivial, all the interactions were local and not too numerous. Such aspects will be investigated in the future.

⁹ <http://www.jolie-lang.org/>

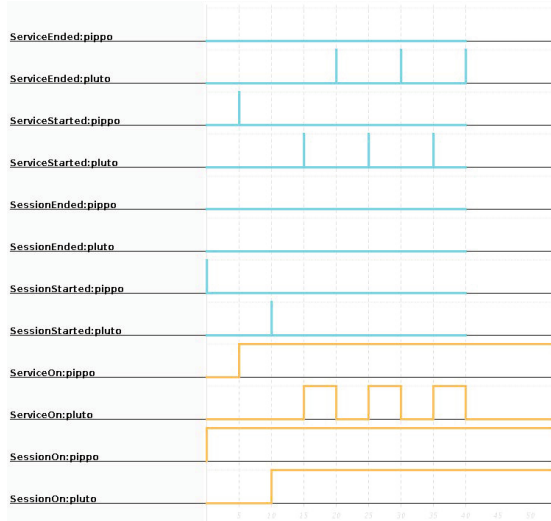


Fig. 8. The graphical output of the service that displays the outcome of the deductive reasoning performed by the EC module when observing the first steps of a Jolie server: starting and ending events for services and sessions are captured (cyan, top) and their effects are reflected on their fluents (light orange, bottom)

5 Conclusions and Future Work

In this paper we have presented our rule based implementation of two incremental EC variants especially suited to be used with other forms of reasoning in a CEP context. To the best of our knowledge, this is the first realisation of this kind. In addition, its strong modular attitude qualifies it to be used in conjunction with other reasoning modules that are blossoming on our reference development platform. In the future, we intend to continue this work and extend it toward monitoring tasks with particular attention to the case of Web services and elders' movements to prevent falls. Other possible future research topics are the support of additional EC variants like the EEC that might be of some help in distributed and heterogeneous domains such as the SOA and augmenting the current EC machinery with abductive and deductive reasoning.

Acknowledgements. Many thanks to Maurizio Gabbrielli, Claudio Guidi and Fabrizio Montesi for helpful discussions about the Jolie framework. We are also grateful for Fabio Ciotoli that has partially contributed with his MSc thesis. Special thanks to the DEIS Depict and EU-FP7 Farseeing projects for partially funding this work.

References

1. Kowalski, R.A., Sadri, F.: Towards a Logic-based Production System Language. Technical report, Department of Computing, Imperial College London (2010)
2. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Upper Saddle River (2003)
3. Simon, H.: Production Systems. The MIT Press (1999)
4. Thagard, P.: Mind: Introduction to cognitive science, 2nd edn. The MIT Press (2005)
5. Kowalski, R.A., Sadri, F.: Programming with logic without logic programming. Technical report, Department of Computing, Imperial College London (2012)
6. Kowalski, R.A., Sergot, M.: A logic-based calculus of events. *New Generation Computing* 4(1), 67–95 (1986)
7. Miller, R., Shanahan, M.: The event calculus in classical logic-alternative axiomatizations. *Electronic Transactions on Artificial Intelligence* 4 (1999)
8. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Stanford University (1968)
9. Sadri, F., Kowalski, R.A.: Variants of the event calculus. In: Proc. ICLP, vol. 95, pp. 67–82 (1995)
10. Shanahan, M.: The Event Calculus explained. *Artificial Intelligence Today*, 409–430 (1999)
11. Mueller, E.T.: Event calculus. *Foundations of Artificial Intelligence* 3, 671–708 (2008)
12. Kowalski, R.A.: Database updates in the event calculus. *The Journal of Logic Programming* 12(1-2), 121–146 (1992)
13. Cervesato, I., Franceschet, M., Montanari, A.: A guided tour through some extensions of the event calculus. *Computational Intelligence* 16(2), 307–347 (2000)
14. Shanahan, M.: Solving the frame problem: a mathematical investigation of the common sense law of inertia. The MIT Press (1997)
15. Chittaro, L., Montanari, A.: Efficient handling of context-dependency in the Cached Event Calculus. In: Proc. of TIME 1994 - International Workshop on Temporal Representation and Reasoning, pp. 103–112 (1994)
16. Chesani, F., Mello, P., Montali, M., Torroni, P.: A logic-based, reactive calculus of events. *Fundamenta Informaticae* 105(1), 135–161 (2010)
17. Shanahan, M.: Robotics and the common sense informatic situation, pp. 684–688 (1996)
18. Shanahan, M.: An abductive event calculus planner. *The Journal of Logic Programming* 44(1-3), 207–240 (2000)
19. Mahbub, K., Spanoudakis, G.: Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience. In: Proceedings. 2005 IEEE International Conference on Web Services, ICWS 2005, pp. 257–265. IEEE (2005)
20. Rouached, M., Fdhila, W., Godart, C.: A semantical framework to engineering wsbpel processes. *Information Systems and E-Business Management* 7(2), 223–250 (2009)
21. Fernandes, A.A.A., Williams, M.H., Paton, N.W.: A logic-based integration of active and deductive databases. *New Generation Computing* 15(2), 205–244 (1997)
22. Cicekli, N.K., Cicekli, I.: Formalizing the specification and execution of workflows using the event calculus. *Information Sciences* 176(15), 2227–2267 (2006)

23. Farrell, A.D.H., Sergot, M.J., Sallé, M., Bartolini, C.: Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems* 14(2-3), 99–129 (2005)
24. Weske, M.: *Business process management: concepts, languages, architectures*. Springer (2010)
25. Ten Teije, A., Miksch, S., Lucas, P.: *Computer-based medical guidelines and protocols: a primer and current trends*, vol. 139. IOS Press (2008)
26. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. *IEEE Internet Computing* 9(1), 75–81 (2005)
27. Weiss, G.: *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press (1999)
28. Forgy, C.L.: RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19(1), 17–37 (1982)
29. Luckham, D.C.: *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley (2001)
30. Allen, J.F.: Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11), 832–843 (1983)
31. Bragaglia, S., Chesani, F., Mello, P., Sottara, D.: A Rule-Based Implementation of Fuzzy Tableau Reasoning. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) *RuleML 2010*. LNCS, vol. 6403, pp. 35–49. Springer, Heidelberg (2010)
32. Bragaglia, S., Chesani, F., Fry, E., Mello, P., Montali, M., Sottara, D.: Event condition expectation (ECE-) rules for monitoring observable systems. In: Palmirani, M. (ed.) *RuleML - America 2011*. LNCS, vol. 7018, pp. 267–281. Springer, Heidelberg (2011)

Complex Reactivity with Preferences in Rule-Based Agents

Stefania Costantini and Giovanni De Gasperis

Dip. di Ingegneria e Scienze dell'Informazione e Matematica (DISIM),
Università di L'Aquila, Coppito 67100, L'Aquila, Italy
{stefania.costantini,giovanni.degasperis}@univaq.it

Abstract. In this paper, we extend our previous work on complex reaction in rule-based logical agents. In particular, we introduce the possibility of defining and exploiting complex preferences for choosing the course of action to undertake in response to external events, also based upon a (simplified) form of modal reasoning and on sequences of past events.

1 Introduction

In past work we developed DALI, a logic agent-oriented language that extends prolog with reactive and proactive features [1,2,3,4] (cf. [5] for a comprehensive list of references). DALI in fact is equipped with “event-condition-action” rules (ECA rules) for defining the behavior of an agent in consequence of perception of external events. A distinguished feature that makes DALI proactive and strongly event-oriented is the *internal events*, i.e., the programmer can indicate internal conditions to be interpreted as events, to which a reaction can be defined. These conditions are checked automatically at a certain frequency, and then treated exactly like external events. The DALI interpreter provides directives that allow a programmer to influence reactivity by indicating at which frequency (if different from default one) events occurrence should be checked, and also since when and until when and/or upon which conditions. We defined management of events which occurred together (with the possibility to customize the time interval defining ‘together’) and priorities between events. In [6] and later in [7] we tackled the issue of complex reactivity in logical agents, by considering the possibility of choosing among different possible reactive patterns by means of simple preferences.

In the meanwhile, event processing (also called CEP, for “Complex Event Processing”) has emerged as a relevant new field of software engineering and computer science [8]. In fact, a lot of practical applications have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions [9,10,11,12] (cf. also the Proceedings of the RuleML Workshop Series). Many products for event processing have appeared on the market, provided by major software vendors and by many start-up companies around the world (see e.g. [11,12] and the references therein). With cloud computing, the effectiveness and usefulness of event processing is even more visible, and a connection of “event pattern languages” with ontologies and with the semantic web is envisaged. Many of the current approaches are declarative and based on rules, and often on logic-programming-like languages and semantics: for instance, [9] is based upon a specifically defined interval-based Event Calculus.

In this paper, we continue and extend the work of [6] by introducing more complex forms of preferences among applicable reactive behaviors. Such preferences can be also defined in terms of “possible worlds” elicited from a declarative description of a current or hypothetical situation, and can depend upon past events, and the specific sequence in which they occurred. To the best of our knowledge, these advancements are orthogonal to the various proposals appeared in the literature, into which they could be possibly merged. The declarative formalism that we use for describing a situation is Answer Set Programming (ASP, cf., [13][14][15][16]), that has proved apt at a variety of complex reasoning tasks (cf., among others, [17][18][19][16] and the references therein), and provides various options for connection to ontologies and semantic web frameworks (cf., e.g., [20] and the references therein).

As we have prototypically implemented the proposed approach using the DALI language [12] (cf. [5] for a complete list of references about DALI), in the rest of the paper we use a sample syntax which is reminiscent of DALI. However, we invite the reader not to consider this syntax as mandatory. In fact, we have tried to design the approach so as to make it applicable to many languages and frameworks. The approach is particularly well-suited for rule-based (even better prolog- or datalog-based) languages. It would not be difficult to transform our syntax into a more standard XML-like form, so as to fit into the area of Semantic Web and Rule Markup Languages (e.g., RuleML [21]). Some of the features of our approach could be in fact usefully integrated into existing approaches to CEP. In this paper we do not propose examples taken from real industrial applications. We instead propose intuitive simple examples that should make it easier to understand the new proposed features. We assume the reader to have basic notions of logic programming and Answer Set Programming.

2 Background

In this section, we recall parts of our previous work that define some basic foundation elements of the proposed approach.

2.1 Declarative Semantics of Logic Agent-Oriented languages

We base our proposal upon the declarative semantic framework introduced in [22], aimed at encompassing approaches to evolving logical agents, by understanding changes determined by external events and by the agent’s own activities as the result of the application of program-transformation functions.

We abstractly formalize an agent as the tuple $Ag = \langle P_{Ag}, E, I, A \rangle$ where Ag is the agent name and P_{Ag} is the “agent program” according to the specific language adopted. E is the set of the external events, i.e, events that the agent is capable to perceive and recognize: let $E = \{E_1, \dots, E_n\}$ for some n . I is the set of internal events (distinguished internal conclusions, that may include desires and intentions): let $I = \{I_1, \dots, I_m\}$ for some m . A is the set of actions that the agent can possibly perform: let $A = \{A_1, \dots, A_k\}$ for some k . Let $\mathcal{Y} = (E \cup I \cup A)$.

In DALI syntax, used below for the examples, atoms indicated with a postfix correspond to events of various kinds. In particular, if p is an atom, pE is an external event, pA is an action and pI an internal event.

According to this semantic account, one will have an initial program $Pe_0 = P_{Ag}$ which, according to events that happen, agent's activities and actions which are performed, will pass through corresponding program-transformation steps (each one transforming Pe_i into Pe_{i+1} , cf. [22]), and thus gives rise to a Program Evolution Sequence $PE = [Pe_0, \dots, Pe_n, \dots]$. The program evolution sequence will imply a corresponding Semantic Evolution Sequence $ME = [M_0, \dots, M_n, \dots]$ where M_i is the semantic account of Pe_i .

Different languages and different formalisms in which an agent can possibly be expressed will influence the following key points: (i) when a transition from Pe_i to Pe_{i+1} takes place, i.e., which are the external and internal factors that determine a change in the agent; (ii) which kind of transformations are performed; (iii) which semantic approach is adopted, i.e., how M_i is obtained from Pe_i .

The semantic account includes an *Initialization step*, where the program P_{Ag} written by the programmer is transformed into a corresponding program Pe_0 by means of some sort of knowledge compilation. In DALI for instance, the initialization step extracts the list of internal and external events, and the control directives that are associated to the program (e.g., for defining priorities among events and frequencies for checking the occurrence of events). In general in fact, Pe_0 can be simply a program (logical theory) or can have additional control information associated to it.

Agents usually record events that happened and actions that they performed. Notice that an agent can describe the state of the world only in terms of its perceptions, where more recent remembrances define the agent's approximation of the current state of affairs. We thus define set Pof current (i.e., most recent) past events, and a set PNV where we store all previous ones. We define the 'history' H of an agent as the tuple $\langle \mathcal{P}, PNV \rangle$, dynamically augmented with new events that happen. In DALI, a past event in \mathcal{P} is in the form $pP : T_i$, where p is an atom corresponding to an event, postfix P stands for 'past' and T_i is a time-stamp indicating when the event has been perceived. In [23] we have defined *Past Constraints*, which allow one to define when and upon which conditions (apart from arrival of more recent versions) past events should be moved into PNV.

Definition 1 (Evolutionary Semantics). *Let Ag be an agent. The evolutionary semantics ε^{Ag} of Ag is a tuple $\langle H, PE, ME \rangle$, where H is the history of Ag , and PE and ME are its program and semantic evolution sequence.*

In [22] the detailed semantic treatment of some basic agent-oriented constructs is provided, in particular that of "condition-action-rules"

$$IF \langle Conditions \rangle DO \langle Actions \rangle$$

that define simple reactivity: whenever the *Conditions* hold, the corresponding *Actions* are performed. Whenever the conditions include external events that have happened, such rules are called "event-condition-action-rules" (ECAs). These rules have been introduced in logic agent-oriented languages since the seminal work of [24].

In our sample syntax, a reactive rule will be indicated with $pE :> Body$ meaning that whenever the external event p is perceived (as said, for the sake of immediate readability external events are indicated with postfix E), the agent will execute *Body*. In what follows, for the sake of clarity we will assume the body to consist just of a

single action. This implies no loss of generality as the extension to a more general form is easily done. In [22], these rules are treated in the initialization step so as to be transformed into an intermediate form then managed by a suitably defined program-transformation step.

2.2 Answer Set Modules

In [7], we have proposed kinds of ASP (Answer Set Programming) modules to be invoked by a logical agents. In particular, one kind is defined so as to allow forms of reasoning to be expressed on possibility and necessity analogous to those of modal logic. In this approach, the “possible worlds” that we consider refer to an ASP program Π and are its answer sets. Therefore, given atom A , we say that A is possible if it belongs to some answer set, and that A is necessary if it belongs to the intersection of all the answer sets.

Precisely, given answer set program Π (also called ‘module’) with answer sets as M_1, \dots, M_k , and an atom A , the *possibility* expression $P(w_i, A)$ is deemed to hold (w.r.t. Π) whenever $A \in M_{w_i}$, $w_i \in \{1, \dots, k\}$. The possibility operator $P(A)$ is deemed to hold whenever $\exists M \in \{M_1, \dots, M_k\}$ such that $A \in M$. Given answer set program Π with answer sets M_1, \dots, M_k , and an atom A , the *necessity* expression $N(A)$ is deemed to hold (w.r.t. Π) whenever $A \in (M_1 \cap \dots \cap M_k)$. Module Π can be implicit (if unique) or explicit, where expressions take the form $P(\Pi, w_i, A)$, $P(\Pi, A)$ and $N(\Pi, A)$ respectively.

Possibility and necessity can possibly be evaluated within a context, i.e., if $E(Args)$ ($E = P$ or $E = N$) is either a possibility or a necessity expression, the corresponding *contextual* expression has the form $E(Args) : Context$ where *Context* is a set of ground facts and rules. $E(Args) : Context$ is deemed to hold whenever $E(Args)$ holds w.r.t. $\Pi \cup Context$, where, with some abuse of notation, we mean that each rule in *Context* is added to Π . The answer set module T where to evaluate an operator can possibly be explicitly specified, in the form: $E(T, Args) : Context$.

In this approach, one is able for instance to define meta-axioms, like, e.g., the following, which states that a proposition is plausible w.r.t. theory T if, say, it is possible in at least two different worlds, given context C :

$$plausible(T, Q, C) \leftarrow P(T, I, Q) : C, P(T, J, Q) : C, I \neq J.$$

3 Complex Reactivity with Preferences in Logical Agents: Past Work, Integrations and Extensions

In [6] we have proposed to employ preferences for defining forms of complex reactivity in logical agents.

The studies of the processes that support the construction or the elicitation of preferences have historically deep roots. In logic, [25] initiated a line of research that was subsequently systematized in [26] and continues nowadays, e.g. in [27] and [28]. Preferences handling in computational logic has been extensively studied too. The reader may refer to [29][30] for recent overviews and discussion of many existing approaches

to preferences that for lack of space we cannot directly mention here. The approach of [31] adopts in particular a form of defeasible logic.

Some of the authors of this paper have proposed approaches to preferences in agents [6] or more generally in logic languages [32][33]. In particular, the approach defined in [32] allows for the specification of various kinds of non-trivial preferences. These preferences follow the quite intuitive principles first formalized in [26], and illustrated at length, e.g., in [27]. The first two principles state that any preference relation is asymmetric and transitive. For simplicity we stick to strict preferences, i.e., if in a certain context one prefers ϕ to ψ , then in the same context one cannot also prefer ψ to ϕ . In our approach, each preference holds in the context of the rule where it is defined. Different (even contrasting) preferences can be expressed (and simultaneously hold) in different rules. The third principle states that preferring ϕ to ψ means that a state of affairs where $\phi \wedge \neg\psi$ holds is preferred to a state of affairs where $\psi \wedge \neg\phi$ holds. The fourth principle states that if one prefers ψ to $(\phi \vee \zeta)$ then (s)he will prefer ψ to ϕ and ψ to ζ . Finally, the last principle states that a change in the world might influence the preference order between two states of affairs, but if all conditions stay constant in the world (“*ceteris paribus*”), then so does the preference order.

We propose an example (reported from [34]) in order to illustrate the approach to preferences in logical agents and languages that we have developed in previous work [6][32][33]. The logic program below defines a recipe for a dessert. The construct *icecream* $>$ *zabaglione* is called a *p-list* (preference list) and states that with the given ingredients one might obtain either ice-cream or zabaglione, but the former is preferred. This is, in the terminology of [26], an “intrinsic preference”, i.e., a preference without a specific reason. In preparing the dessert, one might employ either skim-milk or whole milk. The p-list *skimmilk* $>$ *wholemilk* \leftarrow *diet* states that, if on a diet, the former is preferred. Finally, to spice the dessert, one would choose, by the *p-set* $\{chocolate, nuts, coconut : less_caloric\}$, the less caloric among chocolate, nuts, and coconut. These are instead instances of “extrinsic preferences”, i.e., preferences which come with some kind of “reason”, or “justification”. Notice that, in an agent, extrinsic preferences may change even non-monotonically as the agent’s knowledge base evolves in time, as the justification can be any conjunction of literals.

$$icecream > zabaglione \leftarrow egg, sugar, (skimmilk > wholemilk \leftarrow diet), \\ \{chocolate, nuts, coconut : less_caloric\}.$$

$$less_caloric(X, Y) \leftarrow calory(X, A), calory(Y, B), A < B. \\ calory(nuts, 2). \quad calory(coconut, 3).$$

In general terms, a rule such as the first one in the above program fragment ‘fires’, i.e. can be applied, when the body is entailed by the present knowledge base. In particular, using skim milk or whole milk is conditioned from ‘diet’ (if both are available), and the preferred outcome is ‘ice-cream’. Consider however that preferences expressed in this rule must be combined to preferences possibly expressed in other rules. There are several politics for doing so, discussed in the above references. In the approach of [35] these politics can be explicitly specified via suitable operators by associating a set of *preference rules* to given program.

The above features can be smoothly incorporated into agent-oriented rule-based languages. In fact, the evolutionary semantics presented in [22] can easily accommodate this kind of preference reasoning.

As a first step towards complex reactivity, in the approach of [6], a disjunction among two or more actions may occur in the body of a reactive rule that specifies the response to an event. Associated to the reactive rule are preferences, which are local to the rule where the disjunction occurs, that establish which action is preferred under which conditions; actions as customary have preconditions, thus in any situation the agent should perform the best preferred feasible action.

As an agent evolves in time and its knowledge changes, preferred choices will change as well. Then, according to the same preference structure an agent will in general prefer differently at different stages of its life.

In our sample syntax, we assume action a to be represented as aA . Actions may have preconditions: the connective $:<$ indicates that a rule defines the precondition of an action. I.e., a precondition rule will be indicated as $qA :< Body$, meaning that the action qA can be performed only if $Body$ is true. We do not cope here with the effective execution of actions, that is left to the language run-time support.

A disjunction (indicated with “|”) of actions may occur in the body of a reactive rule. Then, a rule $pE :> q1A | q2A, Body$. means that in reaction to pE the agent may perform indifferently either action $q1A$ or action $q2A$ and then it executes $Body$ [1].

Preferences among actions are defined in *preference condition expressions* associated to a reactive rule. Then, a rule $pE :> q1A | q2A, Body :: q2A > q1A :- Conds$. means that in reaction to pE the agent may perform either action $q1A$ or action $q2A$, but action $q2A$ is preferred over action $q1A$ provided that $Conds$ holds. I.e., if $Conds$ is not verified then the preference is not applicable, and thus any of the actions can be indifferently executed. In general, a disjunction may contain several actions, and several preference condition expressions can be expressed, by means of preference lists (seen before).

These expressions define a *partial order* among actions, where preferences are transitively applied and actions that are unordered can be indifferently executed. In our approach preferences are applied on *feasible* actions. I.e., the partial order among actions must be re-evaluated at each step of the agent life where a choice is possible, according to the preconditions of the actions. The preferred actions at each stage are those that can actually be performed and that are selected by the preference partial order.

Example 1. Consider a person who receives an invitation to go out for dinner. She would prefer accepting the invitation rather than refusing, provided that the invitation comes from nice people. She is able to accept if she has time. The invitation is an *external event* that reaches the agent from her external environment. Accepting or refusing constitutes the *reaction* to the event, and both are actions. One of the actions (namely, accepting) has preconditions, i.e., to have time, and the money to pay for the restaurant. In our sample syntax, an agent program fragment formalizing this situation may look as follows.

¹ Notice that, as reactive rules perform forward reasoning, the part after the $:>$ is a ‘consequence’ of the head and not vice versa. Procedurally, any of $q1A, q2A, Body$ might fail.

```
invitation_dinnerE :> acceptA | refuseA ::
    acceptA > refuseA :- nice_people_inviting.
acceptA :< have_time, have_money.
```

Notice that what the agent will do is not known in advance, as the agent evolves in time: the invitation may arrive at a stage of the agent operation when time and money are available, and then the preferred action is chosen. If instead the invitation arrives when there are no resources for accepting, then the agent will have to refuse.

Consider instead a person who receives an invitation to a boring work meeting. She will prefer (unconditionally) to refuse. However, she cannot do that if she does not have an excuse to present. As we can see, preference among the same actions varies according to the context. Also, if the preconditions of a preferred action are not verified, a less preferred one will have to be performed.

```
invitation_meetingE :> acceptA | refuseA ::
    refuseA > acceptA.
refuseA :< acceptable_excuse.
```

In [6], semantics of reaction with preference is provided as a suitable customization of the evolutionary semantics seen before. In particular, the program evolution step related to reaction with preferences does the following: (i) evaluates actions preconditions to identify feasible actions; (ii) solves the optimization problem defined by preference expressions (where any action among equally preferred ones can be indifferently selected); (iii) replaces the reactive rules with preferences with a plain reactive rule where a best preferred action occurs; (iv) performs as usual for reactive rules.

An immediate extension that can be done to [6] is to allow all forms of p-lists and p-sets as seen above to occur in preference expressions.

A limitation of the approach is that possible actions have to be listed explicitly, while in different situations different action sets should be considered by the agent. For instance, if one is at home in bed with a flu, (s)he cannot possibly accept an invitation, whatever the preference. A distinction between preconditions of actions and general feasibility/unfeasibility is in order. In fact, for the sake of elaboration-tolerance [36] it is impractical and sometimes impossible to specify in advance all circumstances that may prevent an action from being feasible. In our opinion it is better to evaluate general feasibility w.r.t. the present situation, and then evaluate specific preconditions.

In order to determine which actions can possibly be performed in reaction to an event in a given situation, in [7] we have introduced *reactive ASP modules* that describe a situation and, triggered by events that have happened, will have answer sets which encompass the possible reactions to these events (e.g., in the above examples, there will possibly be an answer set containing *acceptA* and another one containing *refuseA*). A reactive ASP module has an input/output interface. The input interface specifies the event(s) that trigger the module. The output interface specifies the actions that the module answer sets (if any) can possibly encompass. Each answer set can give as output one or more actions, that can be selected either indifferently or according to preferences/priorities. So, there will be at least as many actions to consider (according to preconditions and preferences) as the number of answer set of M . A possible new form of the above rule can be for instance:

$invitation_meetingE :> action(M) ::$
 $refuseA > acceptA.$
 $refuseA :< acceptable_excuse.$

where after the $>$ it is stated that any of the actions that are outputs of the ASP module M can be possibly performed, given the local preconditions and the preferences. Module M must be updated according to the present *context*. Say, e.g., that in at present the meeting is important but one has flu: then, the outcome instead of *acceptA* or *refuseA* could be for instance *postponeA*.

It can be interesting to define reaction in terms of meta-statements involving possibility and necessity w.r.t. module M . For instance, in the following example the reaction to event *evE* can be either any action produced by M as a possible reaction, or a *necessary* action, i.e., an action that belongs to all the answer sets of M . The latter is preferred in a critical situation.

$evE :> necessary(M) | action(M).$
 $necessary(M) > action(M) :- critical_situation.$

4 Complex Reactivity with Preferences in Logical Agents: Modal Preferences

Often, the kind of reaction that an agent might prefer to pursue in consequence of a certain event is related to the objectives that the agent intends to reach, or to conditions that the agent would like to fulfill. In order to introduce new more involved forms of reactions possibly based on commonsense and non-monotonic reasoning, or on “local” planning about the consequences that selected reaction strategies may have, we improve the forms of preferential reasoning introduced so far.

To this aim, by drawing inspiration from the work of [28], we define further extensions to expressing preferences in logical agents². In particular, referring to agents, [28] introduces a concept of complex preference where an agent prefers ϕ over ψ if, for any “plausible” (i.e., presumably reachable) world where ψ holds, there exists a world which is *at least as good* as this world and *at least as plausible* where ϕ is true. [28] writes $B(\psi \rightarrow \langle H \rangle \phi)$ where H is a new modality, and the reading is “Hopefully ϕ ”. Semantically: if \mathcal{M} is a preference model encompassing a set of worlds W and $s, t \in W$, \leq is a reachability relation meaning “at least as plausible” and \preceq a preference relation, we have that:

$$\mathcal{M}, s \models H \phi \text{ iff for all } t \text{ with both } s \leq t \text{ and } s \preceq t : \mathcal{M}, t \models \phi$$

ASP modules are a good tool for redefining, extending and implementing the H operator in practical languages. In particular, to stay within a logic programming computationally affordable setting, we do not fully represent reachability and preferability among worlds. Rather, “possible worlds” will be interpreted as the answer sets of a suitable ASP module Π representing the situation at hand. So, all the answer sets of Π are equally reachable. Preferability among worlds is explicitly stated as a property that the

² A preliminary version of part of the material presented in this section was presented in [37].

agent desires to hold. Thus, we define the H operator in terms of the aspect that makes a world in which ϕ holds preferable. In fact, we want to express preferences such as the following, where one may choose to prefer a certain food rather than another one, in the hope that the preferred food is good for health:

$$eat(pasta) > eat(meat) : H(healthy)$$

Here, ψ is $eat(meat)$ while ϕ is $eat(pasta)$, and I prefer a world where ϕ holds because I hope that *healthy* (speaking of myself) will hold as well. Below we indicate this third novel element that we introduce, i.e., the hoped-for reason for preference, with ξ .

As a basic step for defining such a preference we define $\phi : H(\xi)$ (or $\phi : H(\Pi, \xi)$ if ASP module is explicitly indicated) meaning that, assuming ϕ , we expect that ξ will hold in some reachable world, that in our setting is an answer set of an ASP module that can be either implicit or explicitly indicated. Thus, ξ is the “reason why” reachable worlds in which ϕ holds are preferred. We can thus define the operator H by a simple adaptation of the contextual possibility operator as introduced in Section 2.2 where ϕ is taken as the context.

Definition 2. *Given an answer set module Π with answer sets M_1, \dots, M_k , an atom ϕ and an atom ξ , the expression $\phi : H(\Pi, \xi)$ is deemed to hold (w.r.t. Π) whenever the contextual possibility expression $P(\Pi, \xi) : \phi$ holds. The answer set module Π can be left implicit if it is unique, thus leading to the simplified form $\phi : H(\xi)$.*

We can now formally define the above preference expression as follows.

Definition 3. *Given atoms A, B, C and answer set module T , the construct $A > B : H(T, C)$ is called an mp-list (modal preference list) meaning that A is preferred to B (i.e., we have the p-list $A > B$) if $A : H(T, C)$ holds. Otherwise, any of A or B can be indifferently chosen.*

A similar but stronger formulation of previous example can be the following, where one chooses to prefer a food that in most situations can “reasonably” expected to procure better health (where as before these situations are interpreted as the answer sets of an underlying ASP module). The operator $maxH$ (“maximal hope”) intuitively means that the former food is preferred if it is most likely to procure good health.

$$eat(pasta) > eat(meat) : maxH(healthy)$$

To define these more involved preferences, the basic operator H must be extended to a form $\phi : H[N](\xi)$ meaning that, given ϕ , the hoped-for property ξ holds in exactly N different possible worlds.

Definition 4. *Given an answer set module Π with answer sets M_1, \dots, M_k , an atom ϕ and an atom ξ , the expression $\phi : H[n](\xi)$ is deemed to hold (w.r.t. Π) whenever there exist $\{v_1, \dots, v_n\}$, $v_i \in \{1, \dots, k\}$ such that $P(v_i, \xi) : \phi$ holds, $i \leq n$, and for every $P(w_i, \xi) : \phi$ which holds, $w_i \in \{v_1, \dots, v_n\}$. By convention, we assume $\phi : H[0](\xi)$ to signify that $\phi : H[n](\xi)$ holds for no n .*

We can now extend Definition 3 so as to compare A and B w.r.t. how often a satisfactory state of affairs can be reached. That is, we compare A and B on the basis of hoped-for condition C . We prefer A over B if we assess that by assuming A it is more plausible to reach C , i.e., C holds in more worlds than it is by assuming B .

Definition 5. Given atoms A, B, C and answer set module T , the construct $A > B : \max H(T, C)$ is called an mmp-list (modal max-preference list) meaning that A is preferred to B (i.e., we have the p-list $A > B$) iff we have $A : H[N_A](T, C)$ and $B : H[N_B](T, C)$, and $N_A \geq N_B$.

The extension to preference lists with more than two elements is straightforward. The “preference condition” C can be generalized to conjunctions. Operators H and $\max H$ can be contextual, in the form:

$$A : \text{Context} : H(T, C) \text{ and } A : \text{Context} : \max H(T, C)$$

where *Context* is a conjunction of rules, each of which to be added to the ASP module T before evaluating H . This allows for expressions such as:

$$A > B : \text{Context} : H(T, C) \text{ or, respectively, } A > B : \text{Context} : \max H(T, C)$$

The aim of introducing contextual operators is expressivity in the practical sense: this formulation in fact allows one to state in when and why one has a certain preference. We can thus propose for instance the following variation of the above example:

$$\text{eat}(\text{fruit_salad}) > \text{eat}(\text{cake}) : \text{diabetes} : \max H(\text{healthy})$$

It can be also useful to introduce a generalized form of p-set, so as to allow for instance for the representation below, which takes the varieties of food generated by $\text{food}(F)$ and the context *diabetes*, and generates a p-list where the various kinds f of foods are ordered according to the degree of healthiness, interpreted as the value of N in expression $f : \text{diabetes} : H[N]\text{healthy}$.

$$\{\text{food}(F), \text{eat}A(F) : \text{diabetes} : H(\text{healthy})\}$$

To define these new expressions, that we call *modal p-sets*, we have to start from their ground version, where atoms $\text{eat}(f)$ are explicitly listed, instead of being generated by $\text{food}(F)$.

Definition 6. A ground modal p-set (*gmp-set*) is an expression of the form:

$$\{A_1, \dots, A_s : B : H(T, E)\}$$

where T is an ASP module, the A_i s are atoms, and B, E are conjunctions of atoms (B possibly empty). This expression stands for the p-list $A_1 > \dots > A_s$ where for each A_j, A_k in this p-list, A_j precedes (is preferred to) A_k iff the following expression holds:

$$A_j > A_k : B : \max H(T, E)$$

Then, to introduce an implicit specification of the A_i 's such as the one adopted in the above example (thus avoiding to list all them explicitly), we resort to a solution similar to the one adopted in Answer Set Programming for weight constraints. In [38], the authors introduce a notion of a conditional literal of the form $l : d$ where l is a literal and the conditional part d is a domain predicate, where the subset of given program defining *domain predicates* consists of *domain rules*, syntactically restricted so that their extension should be relatively efficiently computable.

Definition 7. A modal p-set (*mp-set*) is an expression of the form:

$$\{p(X_1, \dots, X_n), q(X_1, \dots, X_n) : B : H(T, E)\}$$

where T is an ASP module, q is a predicate (possibly defining an action), p is a domain predicate, X_1, \dots, X_n are variables, B is a conjunction of atoms not involving the X_i s and involving terms Y_1, \dots, Y_v , $v \geq 0$ and E is a conjunction of atoms possibly involving the X_i s and the Y_j s. This expression stands for the *gmp-set*

$$\{A_1, \dots, A_s : B : H(T, E)\}$$

where the A_i s are all the possible ground atoms of the form $q(t_1, \dots, t_n)$ such that $p(t_1, \dots, t_n)$ holds.

Modal p-sets allow for the definition of a variety of useful statements and meta-statements for complex reaction. In the following example for instance, in facing a danger the agent has to choose between screaming, running or phoning to the police. The choice will be done that, in the present situation, provides the best expectation of a happy ending of the adventure. This is obtained via a ground modal p-set.

dangerE :>
 $\{call_policeA, runA, screamA : H(safe)\}.$
call_policeA :< *have_phone*.

The following example states that if a baby is hungry one should feed the baby with attention to healthy food.

baby_is_hungryE :>
 $\{food(F), give_food_to_babyA(F) : : H(healthy)\}.$

It is important to emphasize that the quality of reactive reasoning obtainable via the H and $maxH$ operators is high, as ASP allows one to declaratively represent many forms of reasoning and commonsense reasoning, and is able, e.g., to model complex forms of configuration and planning and of reasoning about resources (cf., among others, [17,18,19,16,39] and the references therein). ASP also provides various options for connection to ontologies and semantic web frameworks (cf., e.g., [20,40,41] and the references therein).

The evolutionary semantics can be customized to manage the above reactive rules by defining a program evolution step that: (i) computes the answer sets of involved ASP modules; (ii) computes preconditions of actions; (iii) computes preferences among feasible actions; (iv) re-writes reactive rule into standard form and treats them accordingly.

5 Complex Reactivity with Preferences in Logical Agents: Conditional Preferences on Event Sequences

There can be situations where the course of actions to be undertaken depends upon what happened in the past, i.e., upon past events. Past events may occur in the preconditions of actions, so this need is partially covered already in standard DALI language. However, sometimes the order and the number of times in which events have happened count, thus another extension is in order. In the following example, one prefers (for the sake of civility) to accept an invitation from nice people if it has been reiterated several times in the past without being accepted. One instead prefers to refuse an invitation from relatives already accepted several times. We apologize because for lack of space we use a simplified syntax that doesn't really allow specific invitations to be coupled to the related acceptance/rejection, so the example just provides an intuitive idea of the envisaged construct.

For event sequences we adapt the syntax of regular expressions (for lack of space we can't formally describe the adapted syntax here, some more detail is provided in [42]).

$$\begin{aligned}
 & \textit{invitation_dinner}E :> \textit{accept}A \mid \textit{refuse}A :: \\
 & \textit{invitation_dinner}P^{E+} \textit{not} \textit{accept}P^A : \\
 & \quad \textit{accept}A > \textit{refuse}A :- \textit{nice_people_inviting}. \\
 & \textit{invitation_dinner}P^{E+} \textit{accept}P^A : \\
 & \quad \textit{refuse}A > \textit{accept}A :- \textit{relatives_inviting}. \\
 & \textit{accept}A :< \textit{have_time}, \textit{have_money}.
 \end{aligned}$$

To manage this case, the evolutionary semantics must provide a program evolution step that evaluates a preference expression only if the specified event sequence matches the agent's knowledge base. If several alternatives are feasible, some politics will be applied so as to select one and put it into play.

6 Concluding Remarks

For lack of space we can't discuss here related work, and in particular the relationship and the possible integration of our approach with the interesting work of [9] that proposes a very comprehensive logical reaction rule language that explicitly considers time intervals. As mentioned however, we believe that the treatment of preferences and the use of ASP modules for reasoning about present situation in terms of what is possible and/or necessary and of objectives that can be reached is new.

Though DALI is fully implemented and has been widely experimented in practical (also industrial) applications (see [43] for a recent application that has won the third prize as best demo), up to now there is no full implementation available for the approach proposed here. Rather, some features have been implemented and some simulated in DALI, mainly by means of the "internal events" construct. We have also implemented in DALI the ASP modules and the related operators. A future aim is that of constructing a full implementation an instance of the proposed framework. In the agent context, we intend to consider KGP [44,45,46], DALI and EVOLP [47] (which are fully-defined and fully-implemented approaches to logical agents) that provide the main elements

and can be exploited in combination in an implementation. However, it would also be very interesting to seek an integration with existing (even commercial) approaches and languages for CEP.

In the perspective of CEP, the present proposal fits into a wider framework where we provide (cf. [37,23,42,48]) and the references therein) dynamic self-checking, mandatory whenever it is not known in advance which events will happen and in which order, and advanced memory management.

Acknowledgements. My mentor Gaetano Aurelio Lanzarone ('Elio' for his friends) died at the age of 66 in October 2011 after a long illness. He has been one of the pioneers of prolog and rule-based automated reasoning in Italy, and initiated me into this fascinating research area. My affection and gratitude to Elio will last forever. This paper is dedicated to him.

References

1. Costantini, S., Tocchio, A.: A Logic Programming Language for Multi-agent Systems. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 1–13. Springer, Heidelberg (2002)
2. Costantini, S., Tocchio, A.: The DALI Logic Programming Agent-Oriented Language. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 685–688. Springer, Heidelberg (2004)
3. Tocchio, A.: Multi-Agent systems in computational logic. PhD thesis, Dipartimento di Informatica, Università degli Studi di L'Aquila (2005)
4. Costantini, S., D'Alessandro, S., Lanti, D., Tocchio, A.: Dali web site, download of the interpreter (2010), <http://www.di.univaq.it/stefcost/Sito-Web-DALI/WEB-DALI/index.php>
5. Costantini, S.: The dali agent-oriented logic programming language: References (2012), <http://www.di.univaq.it/stefcost/info.htm>
6. Costantini, S., Dell'Acqua, P., Tocchio, A.: Expressing preferences declaratively in logic-based agent languages. In: Proc. of Commonsense 2007, the 8th International Symposium on Logical Formalizations of Commonsense Reasoning. AAAI Press (2007); Event in honor of the 80th birthday of John McCarthy
7. Costantini, S.: Answer Set Modules for Logical Agents. In: de Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.) Datalog 2010. LNCS, vol. 6702, pp. 37–58. Springer, Heidelberg (2011)
8. Chandy, M.K., Etzion, O., von Ammon, R.: 10201 Executive Summary and Manifesto – Event Processing. In: Chandy, K.M., Etzion, O., von Ammon, R. (eds.) Event Processing, Number 10201 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2011)
9. Paschke, A., Kozlenkov, A.: Rule-Based Event Processing and Reaction Rules. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 53–66. Springer, Heidelberg (2009)
10. Etzion, O.: Event processing - past, present and future. Proceedings of the VLDB Endowment, PVLDB Journal 3(2), 1651–1652 (2010)
11. Paschke, A., Vincent, P., Springer, F.: Standards for Complex Event Processing and Reaction Rules. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 128–139. Springer, Heidelberg (2011)

12. Vincent, P.: Event-Driven Rules: Experiences in CEP. In: Olken, F., et al. (eds.) *RuleML - America 2011*. LNCS, vol. 7018, p. 11. Springer, Heidelberg (2011)
13. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, pp. 1070–1080. The MIT Press (1988)
14. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 365–385 (1991)
15. Marek, V.W., Truszczyński, M.: Stable logic programming - An alternative logic programming paradigm. In: *25 years of Logic Programming Paradigm*, pp. 375–398. Springer (1999)
16. Gelfond, M.: Answer sets. In: *Handbook of Knowledge Representation*. Elsevier (2007)
17. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
18. Leone, N.: Logic Programming and Nonmonotonic Reasoning: From Theory to Systems and Applications. In: Baral, C., Brewka, G., Schlipf, J. (eds.) *LPNMR 2007*. LNCS (LNAI), vol. 4483, p. 1. Springer, Heidelberg (2007)
19. Truszczyński, M.: Logic Programming for Knowledge Representation. In: Dahl, V., Niemelä, I. (eds.) *ICLP 2007*. LNCS, vol. 4670, pp. 76–88. Springer, Heidelberg (2007)
20. Eiter, T., Brewka, G., Dao-Tran, M., Fink, M., Ianni, G., Krennwallner, T.: Combining Nonmonotonic Knowledge Bases with External Sources. In: Ghilardi, S., Sebastiani, R. (eds.) *FroCoS 2009*. LNCS, vol. 5749, pp. 18–42. Springer, Heidelberg (2009)
21. Boley, H.: The RuleML Family of Web Rule Languages (Revised Selected Papers). In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) *PPSWR 2006*. LNCS, vol. 4187, pp. 1–17. Springer, Heidelberg (2006)
22. Costantini, S., Tocchio, A.: The DALI Logic Programming Agent-Oriented Language. In: Alferes, J.J., Leite, J. (eds.) *JELIA 2004*. LNCS (LNAI), vol. 3229, pp. 685–688. Springer, Heidelberg (2004)
23. Costantini, S.: Defining and maintaining agent’s experience in logical agents. In: *Proc. of the Seventh Latin American Workshop on Non-Monotonic Reasoning, LANMR 2011*, vol. 804, pp. 151–165 (2011); also in the *Informal Proc. of the LPMAS “Logic Programming for Multi-Agent Systems” Workshop at ICLP 2011*
24. Kowalski, R., Sadri, F.: Towards a Unified Agent Architecture that Combines Rationality with Reactivity. In: Pedreschi, D., Zaniolo, C. (eds.) *LID 1996*. LNCS, vol. 1154, pp. 135–149. Springer, Heidelberg (1996)
25. Hallden, S.: On the logic of better. *Library of Theoria*, No. 2 Lund: Library of Theoria. Cambridge University Press (1957)
26. von Wright, G.H.: *The logic of preference*. Edinburgh University Press (1963)
27. van Benthem, J., Girard, P., Roy, O.: Everything else being equal: A modal logic for ceteris paribus preferences. *J. Philos. Logic* 38, 83–125 (2009)
28. Liu, F.: Von Wright “the logic of preference” revisited. *Synthese* 175(1), 69–88 (2009)
29. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence* 20(12), 308–334 (2004)
30. Brewka, G., Niemelä, I., Truszczyński, M.: Preferences and nonmonotonic reasoning. *AI Magazine* 29(4) (2008)
31. Dastani, M.M., Governatori, G., Rotolo, A., van der Torre, L.W.N.: Programming Cognitive Agents in Defeasible Logic. In: Sutcliffe, G., Voronkov, A. (eds.) *LPAR 2005*. LNCS (LNAI), vol. 3835, pp. 621–636. Springer, Heidelberg (2005)
32. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of Algorithms in Cognition, Informatics and Logic* 64(1) (2009)

33. Costantini, S., Formisano, A.: Weight Constraints with Preferences in ASP. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 229–235. Springer, Heidelberg (2011)
34. Costantini, S., Formisano, A.: Augmenting weight constraints with complex preferences. In: Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAI Spring Symposium, USA. AAAI Press (2011)
35. Brewka, G.: Complex preferences for answer set optimization. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004), Whistler, Canada, June 2-5, pp. 213–223 (2004)
36. McCarthy, J.: Elaboration tolerance. In: Proc. of Common Sense 1998 (1998), <http://www-formal.stanford.edu/jmc/elaboration.html>
37. Costantini, S., Dell'Acqua, P., Pereira, L.M., Toni, F.: Meta-axioms and complex preferences in evolving logical agents. In: Proc. of 15th Portuguese Conference on Artificial Intelligence (2011); also in the Informal Proc. of the LPMAS "Logic Programming for Multi-Agent Systems" Workshop at ICLP 2011
38. Simons, P., Niemelä, I., Sooinen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2), 181–234 (2002)
39. Costantini, S., Formisano, A.: Answer set programming with resources. *Journal of Logic and Computation* 20(2), 533–571 (2010)
40. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. *Artif. Intell.* 172(12-13), 1495–1539 (2008)
41. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and ontologies for the semantic web. In: Baroglio, C., Bonatti, P.A., Matuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 1–53. Springer, Heidelberg (2008)
42. Costantini, S.: Temporal meta-axioms in logical agents (submitted)
43. Bevar, V., Muccini, H., Costantini, S., Gasperis, G.D., Tocchio, A.: The DALI Logic Programming Agent-Oriented Language. In: Proc. of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems. AISC. Springer (in press, 2012), Paper and demo
44. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A.C., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP Model of Agency for Global Computing: Computational Model and Prototype Implementation. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS (LNAI), vol. 3267, pp. 340–367. Springer, Heidelberg (2005)
45. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. ECAI 2004 (2004)
46. Stathis, K., Toni, F.: Ambient Intelligence Using KGP Agents. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 351–362. Springer, Heidelberg (2004)
47. Alferes, J.J., Brogi, A., Leite, J., Moniz Pereira, L.: Evolving Logic Programs. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 50–61. Springer, Heidelberg (2002)
48. Costantini, S.: Memory, experience and adaptation in logical agents (submitted)

The Pragmatic Web: Putting Rules in Context

Hans Weigand¹ and Adrian Paschke²

¹ Tilburg University, The Netherlands
H.Weigand@uvt.nl

² Freie Universität Berlin, Germany
paschke@inf.fu-berlin.de

Abstract. The Internet is more than a web of computers and more than a web of documents. From a pragmatic point of view it is interesting what people *do* with the Internet and how. Actions and events have a meaning in the context of a process or practice as enveloping a set of shared norms. The norms apply to behavior, but also to interpretation and evaluation, and can be represented and implemented using rule-based systems.

Keywords: pragmatics, smart computing, context, norms.

1 Introduction

Traditionally, the study of signs (semiotics) distinguishes three components: syntax, semantics and pragmatics, where semantics deals with the meaning of symbols and pragmatics with their context of usage. Pragmatics in the sense of dealing with meaning in context is of increasing importance once IT applications evolve from context-free programs to highly context-sensitive devices.

Basically, the Internet is a network infrastructure that allows uniquely identified machines to interact worldwide. The Pragmatic Web [17] can be defined as a particular view on the Internet as a platform of communication and coordination [19]. How does the Internet effectively support coordination? To answer this question we need to go beyond a data exchange view of communication. What do people try to achieve when they send messages or write notes? Fig. 1 shows a conceptual (service-oriented) framework presented in [19] to connect the Internet technology, the coordination it implements, the real-world processes coordinated and their economic meaning as value networks.

Viewing the Internet as a platform for coordination, we can distinguish the “what” and the “how” of coordination. Central to the “what” are events in the “real world”, such as the delivery of a book, a money transfer or the establishment of a friendship. To coordinate events, human agents exchange messages that count as commitments, such as in a book ordering at Amazon, an online bank transaction or a Facebook friendship request/acceptance. In a narrow sense, “commitment” is restricted to promises or appointments preceding the event to be executed [12,5]. In a broader sense, it refers to all intentional stances that human agents make in the context of their mutual interactions, and can include, for instance, the support of a factual statement (“it is

raining”) or evaluative statement (“this book is cool”). More or less directly, these broader commitments also play a role in the coordination of the mutual behavior. When we talk about the “real world”, this does not exclude the Internet itself. For instance, when human agents coordinate a virtual meeting or an online game. Figure 1 goes one step further by distinguishing the physical event as such (happening in time/space) from the value it has for humans, in particular the economic value (but it might be taken more broadly). The value aspect highlights the “why” of the coordination. Value webs or value networks [19] are a way of modeling the value exchanges.

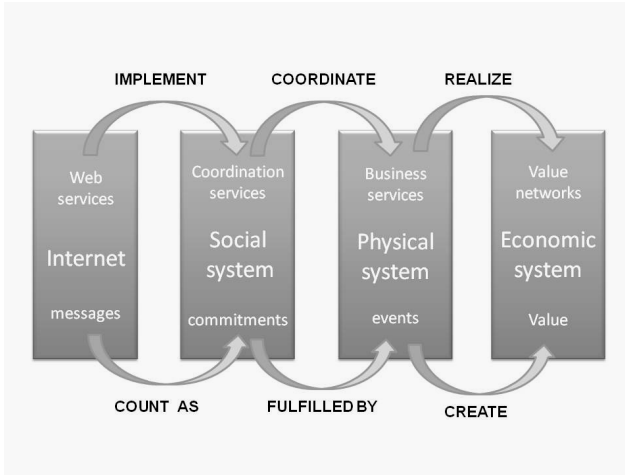


Fig. 1. The Internet as a platform of communication and coordination [19]

The objective of this paper is to clarify the relationship between Pragmatic Web and rule-based systems. In section 2, we will first introduce the central pragmatic notion of context and identify the relationship between context and rules. In section 3, another central pragmatic notion, events, is introduced in the context of Smart Computing. Flexible event-processing requires actable rule representations and rule enforcement mechanisms. We finish with a short conclusion.

2 Pragmatic Web as Web of Contexts and Situations

The Pragmatic Web is a perspective; it is not a web besides the WWW or Semantic Web. Considering the Internet as a platform of communication and coordination means that messages are not viewed in isolation but in the context of an interaction. Although messages can be interpreted to some extent in a context-free manner (the “literal” meaning), this does never exhaust their meaning. We also need to know (or design) its relationship with other messages, and actions, in the context of use.

2.1 Context and Community

According to De Moor [13], ‘The Web’ combines a Syntactic, a Semantic, and a Pragmatic web.

The Syntactic Web consists of interrelated syntactic information resources, such as documents and web pages linked by HTML references. These resources describe many different domains. Each web page has a unique identifier.

The Semantic Web consists of a collection of semantic resources about the Syntactic Web, mainly in the form of ontologies. The ontologies contain semantic networks of concepts, relations, and rules that define the meaning of particular information resources. Ideally, each concept or object has a unique identifier.

The Pragmatic Web, according to De Moor, consists of a set of pragmatic *contexts* of semantic resources. A pragmatic context consists of a common context and a set of individual contexts. A common context is defined by the common concepts and conceptual definitions of interest to a community, the communicative interactions in which these concepts are defined and used, and a set of common context parameters (relevant properties of concepts, joint goals, communicative situation, and so on). Each community member also has an individual context, consisting of individual concepts and definitions of interest and individual context parameters. In the line of the previous, we can say that ideally, each context has a unique identifier to start with.

This approach is in line with the Pragmatic Web Manifesto [17] that states: “The vision of the Pragmatic Web is thus to augment human collaboration effectively by appropriate technologies, such as systems for ontology negotiations, for ontology-based business interactions, and for pragmatic ontology-building efforts in communities of practice. In this view, the Pragmatic Web complements the Semantic Web by improving the quality and legitimacy of collaborative, goal-oriented discourses in communities”. Some would remark that the notion of context can be developed as well within the Semantic Web approach, and in fact has been in recent years, e.g. [1,3]. However, although the representation of context objects does not need to differ fundamentally from other objects, their status is quite different. The pragmatic perspective is interested in how communicative actions get meaning in a context and let it evolve.

De Moor’s approach is also consistent with pragmatism as a research paradigm [7] in IS that takes a central object of study the “practice”. A *practice* is a meaningful ensemble of actors, their different actions, interrelated material, linguistic and institutional elements [16]. A practice is shaped by humans as an organized, artificial and continually evolving arrangement. Practices are at the same time stable (in following institutions and routines) and changing. According to the semiotic framework of Stamper [6], these practices are basically fields of shared *norms*. The norms can apply to behavior (prescriptive, constraining), but there are also cognitive norms that apply to interpretation and evaluation, and norms that apply to the communication, e.g. protocols. Nowadays, norms can be represented and implemented by means of *rule-based systems*, as far as they are explicit (see e.g. Deontic RuleML and Legal RuleML).

2.2 Putting Rules in Context

The Semantic Web builds upon XML as the common machine-readable syntax to structure content and data, upon RDF as a simple language to express property relationships between arbitrary resources (e.g., objects or topics) identified by URIs, and ontology languages such as RDFS or OWL as a means to define rich vocabularies (ontologies) which are then used to precisely describe resources and their semantics. In the recent years, the Linked (Open) Data initiative of Semantic Web created a fast growing cloud of semantically enriched linkages of structured data sources on the web. Together with metadata vocabularies such as Dublin Core, vCard, Bibtext, FOAF and SIOC and further ontologies this prepares an infrastructure to share the relevant information and its knowledge meaning between distributed self-autonomous semantic agents and loosely coupled semantic Web-based services and tools. Rule markup and modeling languages such as RuleML [24] and W3C RIF¹ are used to represent the decision logic and behavioral logic of such agents and to interchange semantic messages (queries, answers, knowledge rules/ontologies, events) between them².

According to Paschke [15], on top of the syntactic and semantic layer, rules play an important role in a pragmatic layer to automatically and contextually transform data into information, interpret this information as (semantic) knowledge, derive new conclusions and decisions from existing knowledge, and behaviorally act according to changed conditions and occurred events.

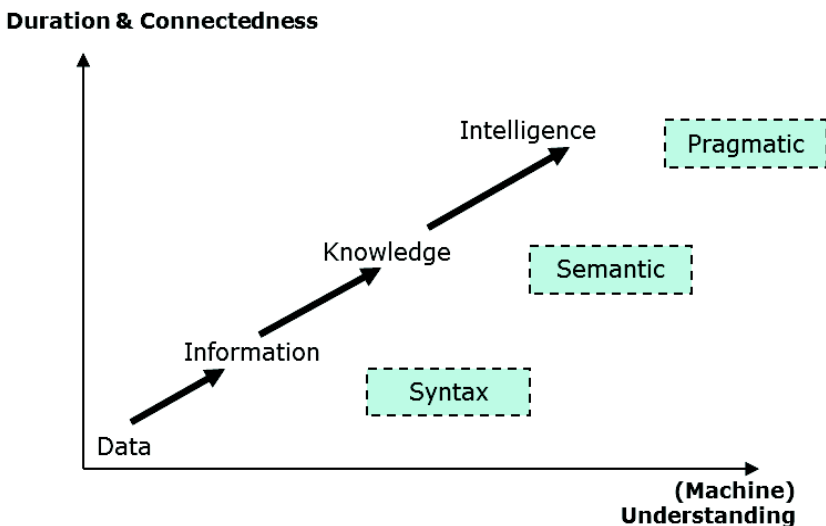


Fig. 2. Semantic-Pragmatic Knowledge Transformation for Machine Intelligence

¹ The W3C RIF recommendation is part of the Semantic Web technology stack. It evolved from RuleML as an XML-based interchange format for rules in the Semantic Web

² Reaction RuleML serves as a standard for representing reaction rules and reactive knowledge (events, actions, messages etc.) [23]

Rules provide a powerful and declarative way to control and reuse the manifold semantically linked meaning representations published on the Semantic Web. Based on an understanding of underlying pragmatic principles this leads to machine understanding for automated “intelligence”. Services and intelligent agents³ can exploit rules to represent their decisions and reactions on how to use knowledge for a particular purpose or goal, including active selection and negotiation about relevant meanings, achievement of tasks, and internal and external reactions on occurred events, changing conditions or new contexts. [25] This extends the Semantic Web to a rule-based Semantic-Pragmatic Web [15] which puts the independent micro-ontologies and domain-specific data into a pragmatic context such as communicative situations, organizational norms, purposes or individual goals and values.

In other words, the Pragmatic Web intends to utilize the Semantic Web with intelligent agents and services that access data and ontologies and make rule-based inferences and autonomous decisions and reaction based on these representations. The focus is on the adequate modeling, negotiation and controlling of the use of the myriad (meta)data and meaning representations of the Semantic Web in a collaborating community of users where the individual meanings as elements of the internal cognitive structures of the members become attuned to each others’ view in a communicative process. This allows dealing with issues like ambiguity of information and semantic choices, relevance of information, information overload, information hiding and strategic information selection, as well as positive and negative consequences of actions.

As a result, this Pragmatic Web becomes more usable in the Social Semantic Web, where it supports the pragmatics in social interactions on the Public Web, as well as in the so called Corporate Semantic Web (CSW)⁴ [34,35,36], where it is used to support the pragmatic application of Semantic Web technologies and Knowledge Management methodologies in corporate environments, e.g., decision support systems (DSS), heterogeneous information systems (HIS) and enterprise application systems (EAS) for distributed human teams and semi-autonomous, agents and IT (web) services: (1) It meaningfully annotates, links, and shares distributed knowledge sources according to common ontologies. (2) It employs rule-based logic for reasoning about source content and metadata. (3) It adds rule-based delegation and integration flow logic to distribute incoming requests towards appropriate virtual (team or organization) members and to collect their responses. By using the Semantic Web as an infrastructure for collaborative networks and by extending it with a rule-based pragmatic and behavioral layer, individuals agents and (Web) services – with their individual contexts, decisions and efforts – can form corporate, educational, or otherwise productive virtual teams or virtual organizations on the Web that have, beside their individual context, a shared context consisting of shared concepts, joint goals and common negotiation and coordination (communication) patterns.

There is currently also a shift from in-house business processes to cross-domain and cross-organizational processes based on an Internet of Services infrastructure with highly flexible and agile IT service management including, e.g., on-demand

³ See e.g. Rule Responder Project: <http://responder.ruleml.org> and [30, 25]. For an overview on rule-based agent systems see [29, 31]

⁴ http://en.wikipedia.org/wiki/Corporate_Semantic_Web

service discovery and coordination based on, e.g. semantic IT service interface descriptions and their SLAs. The Semantic-Pragmatic Web of Agents enables such agile business and service management by providing support for semantic business process management (SBPM) [26], where additional semantic knowledge is used by agents to automate the dynamic (choreography-style) coordination and execution in the context of additional semantic policies and contracts such as Rule-Based Service Level Agreements (RBSLA) [27]. The agent model allows for a semantic and pragmatic abstraction from the underlying services.

3 Pragmatic Web as a Web of Events

The notion of “event” is becoming more and more important in Information Systems and in Business [20]. Real-Time-Enterprise (RTE) takes the role of timeliness to its logical extreme: zero latency, that is, all parts of the enterprise can respond to events as soon as they become known to any one part of the enterprise or extended enterprise. The Event-Driven Architecture (EDA) can support increased agility: to respond to exceptions and unanticipated events at any time, even when business processes are already under way. Complex Event Processing (CEP), often related to as Business Activity Monitoring (BAM), includes tools that monitor the events in the enterprise and are not only able to aggregate data into higher-level complex events but also to detect unusual event patterns that may need an alert [11]. Current standardization efforts are under way in the Event Processing Technical Society (EPTS)⁵, e.g. on a common event processing glossary and vocabulary as well as reference architectures and design patterns [28]. Besides many existing ontologies for events, time, space etc. and industry standards for event communication, there have been also many different approaches for rule-based event processing and reaction rule languages. [33] For an overview on standardizations see [32]. While standards such as OMG Production Rules Representation (UML modeling for production rules) and W3C RIF Production Rules Dialect (Semantic Web interchange format for production rules) focus on production rules, Reaction RuleML [23] is an overarching standard for all types of reaction rules, including rule-based CEP.

Furthermore, we are currently witnessing major steps towards realization of the pervasive and ubiquitous computing vision of Weiser [21]. IT is getting smaller and smaller and increasingly becomes invisible, attached to objects and humans, for example, in the form of an RFID chip. When such objects or humans meet, temporary connections are set up and executed. For example, containers get a unique RFID and when they pass the gate, this automatically triggers the generation of an “enter” event in the Information System. This event may trigger a message event (perhaps via the truck’s navigation system) to be passed to the driver in which he is informed about the exact location where the container is to be put, and may trigger internal tracking & tracing messages to update the information about the container as it is available to all parties involved. At the software level, pervasive computing requires that objects have

⁵ <http://www.ep-ts.com/>

unique identities. This is slowly becoming possible now with the use of EPCIS (electronic product codes, developed for industry and logistics) and, more generally, the Internet of Things [2].

3.1 Smart Computing

The two developments sketched above converge into what is called Smart Computing (Fig. 3). Events are picked up by monitoring functions, aggregated, evaluated and responded to [4]; all these activities are steered by rules. Auditability is a cross-cutting concern. To date, Service-Oriented Architecture (SOA) is the de facto standard in software systems [14]. SOA relies on request/response mechanisms using standards such as WSDL and SOAP. Conceptually, this contrasts with EDA that typically promotes a decoupled communication such as a publish/subscribe mechanism [10]. Technically, a SOA can be built on top of an EDA by modeling the service request and its reply as two different events and ensuring that the reply is triggered by the request. However, there is still the question how to relate the two perspectives on the conceptual level. There are a few recent proposals in this direction, e.g. SOEDA [22]. SOEDA (Service-Oriented Event-Driven Architecture) allows the transformation of event-centric business process notations (EPC) to service-centric execution models (BPEL). According to [10], an event-driven approach should supplement the SOA approach in ubiquitous enterprises.

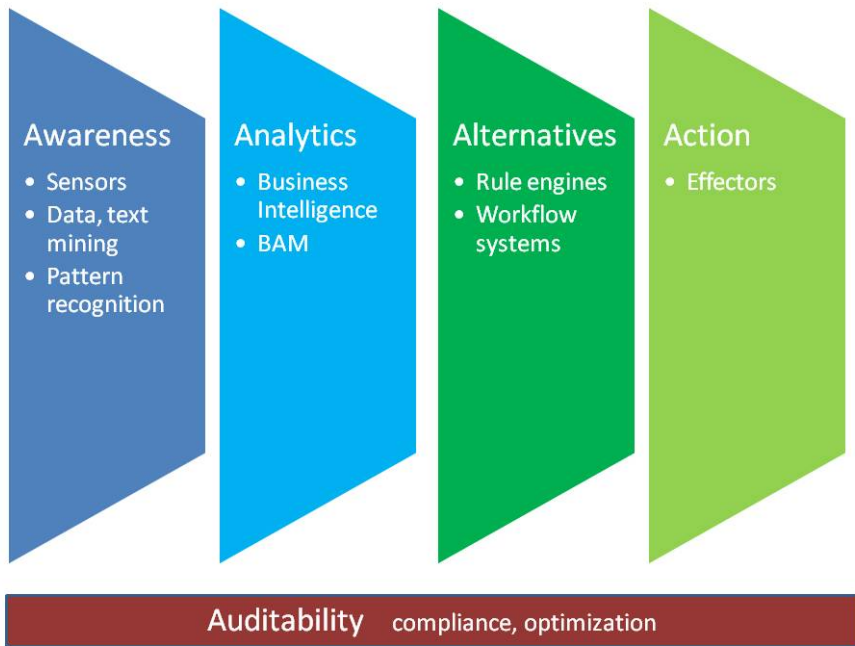


Fig. 3. The 5 A's of Smart Computing, after Forester Inc [4]

3.2 The Pragmatics of Event-Driven Processes

As stated above, pragmatics in the sense of dealing with meaning in context is of increasing importance once IT applications change from context-free programs to highly context-sensitive devices, as in smart computing. Among others, this creates a need for *context* data management [18]. The coupling between data (sensors) and action (effectors) is much tighter than in traditional Information Systems in the office that only interact with the world via the interface of office people. Pragmatics has also to do with the *social* aspect of communication. The social aspect of event-driven processes may seem to be less prominent than traditional workflows running in the office. There are certainly human issues, e.g. privacy concern, that are essential. However, where traditionally Information Systems have been useful tools in coordinating organizational behavior driven by humans, in the ubiquitous computing vision this behavior seems to be driven less and less by humans. To wit, it is no longer the human agent who does the shopping or the purchase ordering, but the fridge or the inventory itself interact with their counter-parts at the supermarket or supplier.

Semantic Web technology can play an important role in the further development of the Internet of Things. Given the enormous amounts of data that will be produced in this kind of applications, a purely syntactic approach is too limiting in the long run. According to [9], two major functions for semantic technologies can be distinguished. First, not surprisingly, they are the basis for the discovery of heterogeneous resources and data integration across multiple domains. Second, they can be used for behavioral control and coordination of the agents representing those resources. In other words, semantic technologies are used both for descriptive specification of the services delivered by the resources and for prescriptive specification of the expected behavior of the resources as well as the integrated system (cf. [16]).

However, not only semantic but also pragmatic aspects such as dealing with context should be taken into account. Economic and legal concerns (e.g. privacy) remain as relevant as before [2]. It may seem that taking humans out of the loop means that social aspects become less relevant. However, the event-driven approach *separates* the social aspect from the physical aspect, but the social aspect, e.g. the authorization structure, still needs to be addressed. The separation observed here in fact indicates a broader evolution. In the area of Electronic Markets, a separation of informational and physical trade processes was observed by Kambil & Van Heck [8]. This separation provides big opportunities for efficiency improvement and increased flexibility. For the internal operations, an authorization structure needs to be described. According to the Smart Computing vision, auditability of the event-driven systems is a general concern. An authorization structure, or authorization governance, in terms of rules, is therefore indispensable.

A recent standardization effort is OASIS LegalRuleML⁶, which enables legal arguments to be created, evaluated, and compared using RuleML representations. This includes, e.g., representation of norms such as deontic norms and other legal norms, legal reasoning on policies and contracts such as SLAs, e.g. in Internet of Service and cloud infrastructures, and representation of law systems (e.g. regulations, patent law).

⁶ <https://www.oasis-open.org/committees/legalruleml/>

4 Conclusion

Rule-based systems are important from a pragmatic perspective, since rules are constitutive of the practices as the contexts of web behavior. At the same time, the notion of context is important for rule development as rules are not global and static, but developed, enforced and continuously adapted by bigger or smaller communities. Hence Pragmatic Web research and rule technology research can beneficially reinforce each other. Particular topics of interest are, e.g., the combination of context management and rule management, a pragmatic rule classification, and policy management.

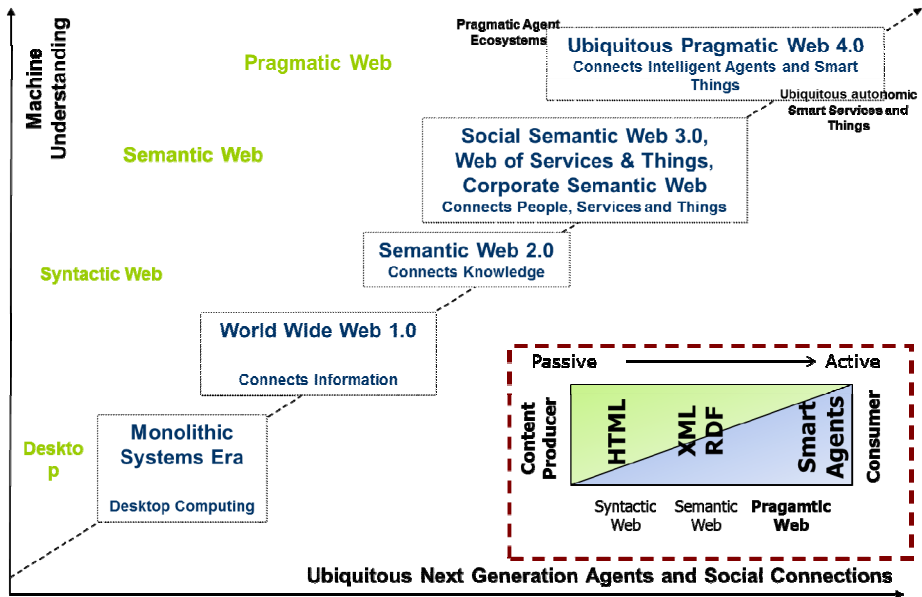


Fig. 4. Vision Ubiquitous Pragmatic Web 4.0

This intelligent behavior of end-user programmable smart agents with their internal and external semantic knowledge and their pragmatic rule-based decision and interactive reaction logic might lead to a transformation of the current passive Web to an ubiquitous active Pragmatic Web 4.0 (see Figure 4), in which an eco-system of pragmatic agents actively supports the human and machine consumers (smart service, smart things) in their desires and needs. This new Web will be inherently event driven with loosely-coupled and decoupled interactions, but with adaptive and real-time behavior.

References

1. Gangemi, A., Presutti, V.: Towards a Pattern Science for the Semantic Web. *Semantic Web* 1(1-2), 61–68 (2010)
2. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A Survey. *Computer Networks* 54, 2787–2805 (2010)

3. Bao, J., Tao, J., McGuinness, D.L., Smart, P.R.: Context Representation for the Semantic Web. In: Web Science Conference, Raleigh, North Carolina, USA (2010)
4. Bartels, A.: Smart Computing Drives The New Era of IT Growth. Forrester Inc. (2009)
5. Dietz, J.: Enterprise Ontology - Theory and Methodology. Springer, Berlin (2006)
6. Filipe, J., Liu, K.: The EDA Model: An Organizational Semiotics Perspective to Norm-Based Agent Design. In: Proc. of the Agents 2000 Workshop on Norms and Institutions in Multi-Agent Systems, Spain (2000)
7. Goldkuhl, G.: The research practice of practice research: theorizing and situational inquiry. *J. Systems, Signs & Actions* 5(1), 7–29 (2011)
8. Kambil, A., van Heck, E.: Reengineering the Dutch flower auctions: A framework for analysing exchange operations. *Inf. Sys. Research* 9(1), 1–19 (1998)
9. Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., Terziyan, V.: Smart semantic middleware for the internet of things. In: Proc. of 5th Int. Conf. on Informatics in Control, Automation and Robotics, ICSO, pp. 169–178 (2008)
10. Kong, J., Jung, J.-Y., Park, J.: Event-driven service coordination for business process integration in ubiquitous enterprises. *Computers & Industrial Engineering* 57, 14–26 (2009)
11. Luckham, D.: The power of events: an introduction to Complex Event Processing. Addison-Wesley (2002)
12. Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: The ActionWorkflow Approach to Workflow Management Technology. *The Information Society* 9, 391–404 (1993)
13. de Moor, A.: Patterns for the Pragmatic Web. In: Dau, F., Mugnier, M.-L., Stumme, G. (eds.) ICCS 2005. LNCS (LNAI), vol. 3596, pp. 1–18. Springer, Heidelberg (2005)
14. Papazoglou, M.P.: Web Services: Principles & Technology. Pearson Education (2008)
15. Paschke, A., Boley, H., Kozlenkov, A., Craig, B.: Rule Responder: RuleML-based Agents for Distributed Collaboration on the Pragmatic Web. In: Proc. 2nd Int. Conf. on the Pragmatic Web. ACM (2007)
16. Schatzki, T.R.: Introduction: Practice theory. In: Schatzki, T.R., Knorr Cetina, K., von Savigny, E. (eds.) *The Practice Turn in Contemporary Theory*, Routledge, London (2001)
17. Schoop, M., Moor, A., Dietz, J.: The Pragmatic Web: A manifesto. *Communications of the ACM* 49(5), 75–76 (2006)
18. Tribowski, C., Goebel, C., Gunther, O.: RFID Context Data Management: The Missing Link to EPCIS-Based Supply Chain Monitoring. In: Proc. PACIS 2009. AIS (2009)
19. Weigand, H., Jayasinghe Arachchig, J.: Value network analysis for the pragmatic web: A case of logistic innovation. In: Proc. I-Semantics 2010. ACM (2010)
20. Weigand, H.: The Pragmatics of Event-Driven Processes. In: Proc. I-Semantics 2011. ACM (2011)
21. Weiser, M.: The Computer for the 21st Century. *Scientific American* 265(3), 94–104 (1991)
22. Wieland, M., Martin, D., Kopp, O., Leymann, F.: SOEDA: A Method for Specification and Implementation of Applications on a Service-Oriented Event-Driven Architecture. In: Abramowicz, W. (ed.) BIS 2009. LNBIP, vol. 21, pp. 193–204. Springer, Heidelberg (2009)
23. Paschke, A., Boley, H., Zhao, Z., Teymourian, K., Athan, T.: Reaction RuleML 1.0: Standardized Semantic Reaction Rules. In: Bikakis, A., Giurca, A. (eds.) RuleML 2012. LNCS, vol. 7438, pp. 100–119. Springer, Heidelberg (2012)
24. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: The Overarching Specification of Web Rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010)

25. Paschke, A., Boley, H.: Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web. *International Journal on Artificial Intelligence Tools* 20(6), 1043–1081 (2011)
26. Paschke, A.: A Semantic Rule and Event Driven Approach for Agile Decision-Centric Business Process Management. In: *Service Wave*, pp. 254–267 (2011)
27. Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. *Decision Support Systems* 46(1), 187–205 (2008)
28. Paschke, A., Vincent, P., Alves, A., Moxey, C.: Tutorial on Advanced Design Patterns in Event Processing. In: *6th ACM International Conference on Distributed Event-Based Systems, DEBS 2012*. ACM, Berlin (2012)
29. Bădică, C., Braubach, L., Paschke, A.: Rule-Based Distributed and Agent Systems. In: Pasche, A. (ed.) *RuleML 2011 - Europe*. LNCS, vol. 6826, pp. 3–28. Springer, Heidelberg (2011)
30. Paschke, A., Boley, H.: Rule Responder - RuleML-Based Pragmatic Web Agents. In: Elci, A., Kone, M., Orgun, M. (eds.) *Semantic Agent Systems*. SCI, vol. 344, pp. 3–23. Springer, Heidelberg (2011)
31. Braubach, L., Pokhar, A., Paschke, A.: Rule-Based Concepts as Foundation for Higher-Level Agent Architectures. In: *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. IGI Publishing (2009) ISBN:1-60566-402-2
32. Paschke, A., Vincent, P., Springer, F.: Standards for Complex Event Processing and Reaction Rules. In: Palmirani, M. (ed.) *RuleML - America 2011*. LNCS, vol. 7018, pp. 128–139. Springer, Heidelberg (2011)
33. Paschke, A., Kozlenkov, A.: Rule-Based Event Processing and Reaction Rules. In: Governatori, G., Hall, J., Paschke, A. (eds.) *RuleML 2009*. LNCS, vol. 5858, pp. 53–66. Springer, Heidelberg (2009)
34. Paschke, A., et al.: Corporate Semantic Web – Technical Reports I-V Technical Report, Freie Universität Berlin, Fachbereich Mathematik und Informatik, http://edocs.fu-berlin.de/docs/receive/FUODOCS_document_000000005877, http://edocs.fu-berlin.de/docs/receive/FUODOCS_document_000000005563, http://edocs.fu-berlin.de/docs/receive/FUODOCS_document_000000011878 (accessed June 2012)
35. Paschke, A., Coskun, G., Heese, R., Luczak-Rösch, M., Oldakowski, R., Schäfermeier, R., Streibel, O.: Corporate Semantic Web: Towards the Deployment of Semantic Technologies in Enterprises. In: Du, W., Ensan, F. (eds.) *Canadian Semantic Web*, pp. 105–131 (August 2010)
36. Coskun, G., Heese, R., Luczak-Rösch, M., Oldakowski, R., Paschke, A., Schäfermeier, R., Streibel, O.: Towards a Corporate Semantic Web. In: *International Conference on Semantic Systems, JUCS Proceedings I-Semantics 2009*, Graz, Austria, September 2–4 (2009)

HARM: A Hybrid Rule-Based Agent Reputation Model Based on Temporal Defeasible Logic

Kalliopi Kravari and Nick Bassiliades

Dept. of Informatics, Aristotle University of Thessaloniki,
GR-54124 Thessaloniki, Greece
{kkravari, nbassili}@csd.auth.gr

Abstract. Multi-agent systems are considered a modern medium of communication and interaction with limited or no human intervention. As intelligent agents are gradually enriched with Semantic Web technology, their use is constantly increasing. To this end, the degree of trust that can be invested in a certain agent is recognized as a vital issue. Current trust models are mainly based on agents' direct experience (interaction trust) or reports provided by others (witness reputation). Though, lately, some combinations of them (hybrid models) were also proposed. To overcome their main drawbacks, in this paper we propose HARM, a hybrid, rule-based reputation model based on temporal defeasible logic. It combines the advantages of the hybrid approach and the benefits of a rule-based reputation modeling approach, providing a stable and realistic estimation mechanism with low bandwidth and computational complexity. Moreover, an evaluation of the reputation model is presented, demonstrating the added value of the approach.

Keywords: Semantic Web, Intelligent Multi-agent Systems, Agent Reputation, Temporal Defeasible Logic, Defeasible Reasoning.

1 Introduction

Intelligent multi-agent systems (MASs) are considered a rather modern medium of communication and interaction with limited or even no human intervention. As, *intelligent agents (IAs)* are gradually enriched with SW technology, realizing the Semantic Web (SW) vision [10], their use is constantly increasing. However, they are also increasingly removing us from the familiar styles of interacting that traditionally rely on some degree of pre-established trust between partners. Moreover, most traditional cues for assessing trust in the physical world are not available, through MASs, anymore. Whenever we, through our agents, have to interact with partners of whom we know nothing, we have to face the challenging task of making decisions involving risk. Thus, the success of an agent, in a MAS, may depend on its ability to choose reliable partners. Nevertheless, a critical issue is now raised: how can an agent trust an unknown partner in an open and thus risky environment?

To this end, a number of researchers were motivated by the understanding that some individuals (agents) may be dishonest. Thus, they have proposed, in different

perspectives, models and metrics of trust and reputation, focusing on estimating the degree of trust that can be invested in a certain agent [23]. In general, reputation is the opinion of the public towards an agent. Reputation allows agents to build trust, or the degree to which one agent has confidence in another agent, helping them to establish relationships that achieve mutual benefits. Hence, reputation (trust) models help agents to decide who to trust, encouraging trustworthy behavior and deterring dishonest participation by providing the mean through which reputation and ultimately trust can be quantified [24]. Currently, computational reputation models are usually built either on *interaction trust* or *witness reputation*, namely, an agent's direct experience or reports provided by others, respectively.

However, both approaches have limitations. For instance, if the reputation estimation is based only on direct experience, it would require a long time for an agent to reach a satisfying estimation level. This is because, when an agent enters an environment for the first time, it has no history of interactions with the other agents in the environment. Thus, it needs a long time to reach a sufficient amount of interactions that could lead to sufficient information. On the other hand, models based only on witness reports could not guarantee reliable estimation as self-interested agents could be unwilling or unable to sacrifice their resources in order to provide reports. Hence, models based only on one or the other approach typically cannot guarantee stable and reliable estimations.

In order to overcome these drawbacks, some researchers, among them the authors of this paper [18], have proposed hybrid models that combine both interaction trust and witness reputation. However, most of hybrid models either have fixed proportion of their active participation in the final estimation or leave the choice to the final user. Although these approaches have significant advantages, sometimes they may lead to misleading estimations. Users may have little or no experience and thus take wrong decisions that could lead to wrong assessments, whereas fixed values provide just generic estimations. Our goal is not to estrange the users from the decision making process, but to help them, and their agents, to make better decisions.

Hence, in order to overcome this drawback and improve the effectiveness of the hybrid approach, we propose a novel incremental solution that combines the advantages of the hybrid approach and the benefits of a rule-based reputation modeling approach, which uses *temporal defeasible logic*. Temporal defeasible logic (TDL) is an extension of *defeasible logic* (DL), developed to capture the concept of temporal persistence [9]. This is an important issue in reputation estimation, as agents may change their objectives at any time. Defeasible logic, on the other hand, is a logic that has the notion of rules that can be defeated, allowing an existing belief to turn false, making it nonmonotonic [20][21]. It is part of a more general area of research, defeasible reasoning, which is notable for its low computational complexity (linear [19]), a property preserved in TDL, too [8].

Moreover, as DL and TDL are nonmonotonic logics, they are capable of modeling the way intelligent agents and humans draw reasonable conclusions from information which falls short of being definitive. These conclusions, despite being supported by the information currently available to the agent, could nonetheless be rejected in the light of new, or more refined, information. In a fundamental sense, nonmonotonic

logics occupy undoubtedly prominent position among the disciplines investigating intelligent reasoning about complex and dynamic situations. Thus, permitting agents to arrive at defeasible conclusions by using TDL, leads to more realistic assessments similar to human reasoning and logic.

In this paper we propose HARM, a hybrid, rule-based reputation model which uses *temporal defeasible logic* and combines both *interaction trust* and *witness reputation*. It is a knowledge-based approach that provides a more intuitive method for non-technical users, letting them draw reasonable and realistic conclusions similar to their own reasoning. Furthermore, HARM provides a stable and reliable estimation mechanism, under a centralized administration authority (agent), overcoming the difficulty to locate witness reports. It is based on well established estimation parameters [3][4], such as information correctness, completeness, and validity, as well as the agent's response time. Additionally, it provides low bandwidth and computational complexity. Finally, an evaluation is presented, demonstrating the added value of the approach.

The rest of the paper is organized as follows. In Section 2, we present a brief overview of temporal defeasible logic. Section 3 presents HARM and its contribution. In Section 4, HARM's evaluation is presented, that demonstrates the added value of the approach. Section 5 discusses related work, and Section 6 concludes with final remarks and directions for future work.

2 Temporal Defeasible Logic

As mentioned, *temporal defeasible logic* (TDL) is an extension of *defeasible logic* (DL). Defeasible logic is a logic that has the notion of rules that can be defeated. It is part of a more general area of research, defeasible reasoning [20][21]. Defeasible reasoning, in contrast with traditional deductive logic, allows the addition of further propositions to make an existing belief false, making it nonmonotonic [14]. One of the main interests in DL, and TDL consequently, is in the area of agents [7]. This is because DL is a nonmonotonic logic and, thus, capable of modeling the way intelligent agents (like humans) draw reasonable conclusions from inconclusive information. This feature, which leads to more realistic conclusions and assessments similar to human reasoning and logic, motivated plenty of researchers. Hence, even in early work performed by Pollock [22], it has been shown that there is a requirement for the addition of a temporal aspect to defeasible reasoning. Much has been done since then; temporal extensions to defeasible logic have already been discussed for agents, leading to temporal defeasible logic.

Obviously, knowledge in TDL is, still, represented as in DL, namely in terms of facts and rules. Facts are indisputable statements, represented either in form of states of affairs (literal and modal literal) or actions that have been performed. Rules describe the relationship between a set of literals (premises) and a literal (conclusion). There are strict rules and defeasible rules. Strict rules take the form $A_1, \dots, A_n \rightarrow B$ ($B:-A_1, \dots, A_n$, in d-POSL syntax [13]). They are rules in the classical sense: whenever the premises are indisputable, then so is the conclusion. Thus, they can be used

for definitional clauses. Defeasible rules take the form $A_1, \dots, A_n \Rightarrow B$ ($B := A_1, \dots, A_n$, in d-POSL syntax). They are rules that can be defeated by contrary evidence. Actually, this is the main concept in TDL (inherited by DL): logic does not support contradictory conclusions, but it tries to resolve conflicts. Hence, in cases where there is some support for concluding A , but there is also support for concluding $\neg A$, no conclusion can be derived unless one of them has priority over the other. This priority is expressed through a superiority relation among rules which defines priorities among them, namely where one rule may override the conclusion of another rule. A special case of conflict is between different positive literals, all derived by different defeasible rules, whereas only one should be derived. “Conflicting literals” are defined through a conflict set and the conflict is resolved through superiorities.

The importance of adding a temporal dimension to rules and literals is undoubted. In TDL there are two types of temporal literals. The first, an expiring temporal literal, consists of a pair $l:t$ that denotes a literal, l , that is valid for t time instances. The second type of temporal literal, a persistent temporal literal, consists of a pair $l@t$ and denotes a literal, l , that is active after t time instances have passed and is valid thereafter. A temporal rule takes the form: $a_1:d_1 \dots a_n:d_n \Rightarrow^d b:d_b$, where that pairs, $e:d$, represent an event identifier, e , and a duration d . The duration of instantaneous events is 0, whereas persistent events have an infinite duration. The \Rightarrow^d indicates the delay between the cause $a_1:d_1 \dots a_n:d_n$ and the effect $b:d_b$. There are many definitions of the semantics of this delay [9]. For instance, the delay may be the delay between the start of the last cause and the beginning of the effect. Hence, temporal rules contain temporal literals. To correctly model the delay between the body and the head, rules are created with the heads of the rules modified to accommodate the delay. For instance, given $(r1) \Rightarrow a@1$ and $(r2) a@1 \Rightarrow^7 b:3$, a literal a is created due to $r1$. It becomes active at time offset 1 but does not cause the head of $r2$ to be fired until time 8. The result b , on the other hand, lasts only until time 10, thereafter, only the fact a remains.

3 HARM

HARM, the proposed model, is a hybrid, rule-based reputation model which uses temporal defeasible logic in order to combine interaction trust and witness reputation. It aims at improving the performance of the hybrid approach by providing a more effective and intuitive decision making mechanism.

3.1 Model Abstract Architecture

First of all, we had to consider the agents’ abilities that should be under evaluation, as an efficient decision making mechanism has to rely on carefully selected data. To this end, after a thorough study of the related literature, four properties were chosen, namely validity, completeness, correctness and response time [3][4].

Validity is the conjunction of sincerity, and credibility. An agent is sincere when it believes what it says, whereas it is credible when what it believes is true in the world. Hence, an agent is valid if it is both sincere and credible. Completeness, on the other

hand, is the conjunction of cooperativity and vigilance. Cooperativity is, actually, defined by duality of sincerity, namely an agent is cooperative when it says what it believes. In the same way, vigilance is defined by duality of credibility, namely an agent is vigilant when it believes what is true in the world. Moreover, correctness refers to an agent's providing services. An agent is correct if its provided service is correct with respect to a specification. Finally, response time refers to the time that an agent needs in order to complete the transaction.

Consider an agent *A* establishing an interaction with an agent *B*; agent *A* can evaluate the other agent's performance and thus affect its reputation. We call the evaluating agent (*A*) *truster* and the evaluated agent (*B*) *trustee*. For some interactions an agent can be both truster and trustee, since it can evaluate its partner while it is evaluated by that partner. After each interaction in the environment, the *truster* has to evaluate the abilities of the *trustee*, and report its rating in terms of *validity*, *completeness*, *correctness* and *response time*. However, that is not enough. The truster has to indicate how confident he/she is for his/her rating and how important the transaction for him/her was. Confidence gives an estimation of the truster's certainty, while transaction importance indicates how critical the transaction was for the truster, allowing us to decide how much attention we should pay on the rating. Taking the above into account the truster's rating value (*r*) in HARM has eight coefficients: (*truster*, *trustee*, *validity*, *completeness*, *correctness*, *response time*, *confidence*, *transaction value*).

In HARM, confidence and transaction values vary from 0 (0%) to 1 (100%) while the rest rating values vary from 0.1 (terrible) to 10 (perfect); $r \in [0.1, 10]$. In order to analyze rating data better, by crossing out extremely positive or extremely negative values, we propose the logarithmical transformation of rating values. The most important feature of the logarithm is that, relatively, it moves big values closer together while it moves small values farther apart. And this is useful in analyzing data, because many statistical techniques work better with data that are single-peaked and symmetric. Furthermore, it is easier to describe the relationship between variables when it is approximately linear. Thus, when these conditions are not true in the original data, they can often be achieved by applying a logarithmic transformation. Hence, each rating is normalized ($r \in [-1, 1] \mid -1 \equiv \text{terrible}, 1 \equiv \text{perfect}$), by using 10 as base. Thus, the final reputation value ranges from -1 to +1, where -1, +1, 0 stand for absolutely negative, absolutely positive and neutral (also used for newcomers), respectively, which means that an agent's reputation could be either negative or positive.

Next, we had to consider another important issue; which experience will be taken into account, direct, indirect (witness) or both. Indirect experience is divided in two categories, reports provided by strangers and reports provided by known agents due to previous interactions. It is well known that using different opinions (ratings) of a large group maximizes the possibility of crossing out unfair ratings, ratings that do not reflect the genuine opinion of the rater, and deals with the discrimination issue. Discrimination means that an agent provides high quality services to one group of partners and low quality to another group of partners. Using both direct and indirect experience could lead to more truthful estimations. Hence, the final reputation value of an agent *X*, required by an agent *A*, is a combination of three coefficients:

$R_{AX} = \{PR_{AX}, KR_{AX}, SR_{AX}\}$. PR_{AX} refers to the agent’s direct experience, KR_{AX} refers to “known” witness ratings, and SR_{AX} refers to the rest witness ratings.

However, sometimes one or more rating categories are missing, for instance, a newcomer has no personal experience and, thus, there are no available ratings (pr_{AX}). To this end, we wish to ground our conclusions in trust relationships that have been built and maintained over time, much as individuals do in real world. A user is much more likely to believe statements from a trusted acquaintance than from a stranger. Thus, personal opinion (AX) is more valuable than strangers’ opinion (SX), as well as it is more valuable even from previously trusted partners (KX). This relationship among the rating categories is presented below, graphically, in Fig. 1.

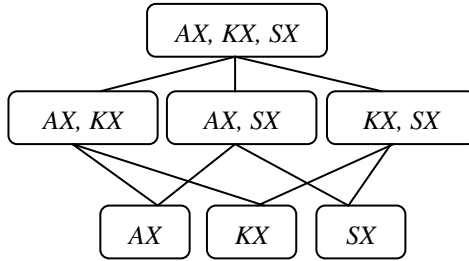


Fig. 1. Superiority relationship among rating categories

Finally, each rating that is going to participate in the estimation, according to the chosen relationship theory (see also subsection 3.2), is used in $R_{AX} = \{PR_{AX}, KR_{AX}, SR_{AX}\}$. More specifically, the final reputation value R_{AX} is a function that combines the transformed ratings for each available category:

$$R_{AX} = \mathfrak{S} \left(PR_{AX}, KR_{AX}, SR_{AX} \right) \tag{1}$$

In addition, whenever an agent requests the reputation value of another agent, maybe he/she wants to determine what is important for him/her. For instance, an agent may be considering response time to be more important (e.g., a 50% on the final value) than correctness (e.g., 30%) or completeness (e.g., 15%). To this end, HARM allows agents to determine weights that will indicate their personal preferences according the ratings’ coefficients. Thus, formula (1) is modified accordingly:

$$R_{AX} = \mathfrak{S} \left[\frac{AVG \left(w_i \times \log \left(pr_{AX}^{coefficient} \right) \right)}{\sum_{i=1}^4 w_i}, \frac{AVG \left(w_i \times \log \left(kr_{AX}^{coefficient} \right) \right)}{\sum_{i=1}^4 w_i}, \frac{AVG \left(w_i \times \log \left(sr_{AX}^{coefficient} \right) \right)}{\sum_{i=1}^4 w_i} \right],$$

$coefficient = \{validity, completeness, correctness, response_time\} \tag{2}$

3.2 Rule-Based Mechanism

Traditionally, humans in real life rely on previous experience and knowledge in order to make decisions and act. Although, available information is mainly inconclusive,

since new and more refined information may come up at any time, humans are able to draw reasonable conclusions from it. Hence, it would be interesting, and probably more effective, to model agents' logic in a human-like (nonmonotonic) logic, since their purpose is to act on behalf of their human users. To this end, knowledge-based (more specifically, rule-based) models, using nonmonotonic reasoning, would let agents to reach the way their human users think and act. Thus, permitting agents to arrive at defeasible conclusions leads to more realistic assessments similar to human reasoning and logic.

In order to present the introduced terms and defeasible rules of our approach in a compact way, we express them in the compact d-POSL syntax [13] of defeasible RuleML [1]. According to our rule-based approach, the trustor's rating (r) is the fact:

*rating(id→rating's_id, trustor→trustor's_name, trustee→trustee's_name,
validity→value₁, completeness→value₂, correctness→value₃,
response_time→value₄, confidence→value₅, transaction_value→value₆).*

Additionally, an example rating provided by agent (A) *trustor* for the agent (B) *trustee* could be:

*rating(id→I, trustor→A, trustee→B, validity→5, completeness→6, correctness→6,
response_time→8, confidence→0.8, transaction_value→0.9).*

Defining the rating values was the first step towards an efficient reputation model, the core of the approach, however, is its decision making mechanism. First, as already mentioned, two of the values, namely, the confidence and the transaction value, allow us to decide how much attention we should pay on that rating. In other words, it is important to take into account ratings that were made by confident trustors, since their ratings are more likely to be right. Additionally, confident trustors, that were interacting in an important for them transaction, are even more likely to report truthful ratings. This assumption led to the following three defeasible rules. These rules define which ratings will be taken into account in the reputation estimation and which not, according to the confidence and the transaction values, but they are not involved in the estimation itself.

r_1 : *count_rating(rating→?idx, trustor→?a, trustee→?x) :=
confidence_threshold(?conf), transaction_value_threshold(?tran),
rating(id→?idx, confidence→?conf_x, transaction_value→?tran_x),
?conf_x ≥ ?conf, ?tran_x ≥ ?tran.*

r_2 : *count_rating(rating→?idx, trustor→?a, trustee→?x) :=
confidence_threshold(?conf), transaction_value_threshold(?tran),
rating(id→?idx, confidence→?conf_x, transaction_value→?tran_x),
?conf_x ≥ ?conf.*

r_3 : *count_rating(rating→?idx, trustor→?a, trustee→?x) :=
confidence_threshold(?conf), transaction_value_threshold(?tran),
rating(id→?idx, confidence→?conf_x, transaction_value→?tran_x),
?tran_x ≥ ?tran.*

$r_1 > r_2 > r_3$

Rule r_1 indicates that if both the truster's confidence and transaction importance are high, according to the user's threshold, then that rating will be counted during the estimation process. Rule r_2 , on the other hand, indicates that even if the transaction value is lower than the threshold, it doesn't matter so much if the truster's confidence is high. Rule r_3 , finally, indicates that if there are only ratings with high transaction value then they should be taken into account. In any other case, the rating should be omitted. Notice that the above rules are defeasible and they all conclude positive literals. However, these literals are conflicting each other, for the same pair of agents (truster and trustee), since we want in the presence e.g. of personal experience to omit strangers' ratings. That's why there is also a superiority relationship between the rules. The conflict set is formally determined as follows:

$$C[\text{count_rating}(\text{truster} \rightarrow ?a, \text{trustee} \rightarrow ?x)] = \\ \{ \neg \text{count_rating}(\text{truster} \rightarrow ?a, \text{trustee} \rightarrow ?x) \} \cup \\ \{ \text{count_rating}(\text{truster} \rightarrow ?a1, \text{trustee} \rightarrow ?x1) \mid ?a \neq ?a1 \wedge ?x \neq ?x1 \}$$

The next step is to decide whose experience will be taken into account, direct, indirect (witness), or both, as explained in section 3.1. Indirect experience, as mentioned, is divided in two categories, reports provided by strangers and reports provided by known agents due to previous interactions. Since we aim at providing a trust estimation much as individuals do in real world, where they built and maintained trust relationships over time, we propose a set of defeasible rules that simulate their decision making process. Hence, HARM, firstly, checks the available ratings categorizing them (rules r_4 to r_7) and then decides which opinion will take into account for the final reputation value R_{AX} (rules r_8 to r_{10}).

$$r_4: \text{known}(\text{agent}_1 \rightarrow ?a, \text{agent}_2 \rightarrow ?y) :- \\ \text{count_rating}(\text{rating} \rightarrow ?id, \text{truster} \rightarrow ?a, \text{trustee} \rightarrow ?y). \\ r_5: \text{count_pr}_{AX}(\text{agent} \rightarrow ?a, \text{truster} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id) :- \\ \text{count_rating}(\text{rating} \rightarrow ?id, \text{truster} \rightarrow ?a, \text{trustee} \rightarrow ?x). \\ r_6: \text{count_kr}_{AX}(\text{agent} \rightarrow ?a, \text{truster} \rightarrow ?k, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id) :- \\ \text{known}(\text{agent}_1 \rightarrow ?a, \text{agent}_2 \rightarrow ?k), \\ \text{count_rating}(\text{rating} \rightarrow ?id, \text{truster} \rightarrow ?k, \text{trustee} \rightarrow ?x). \\ r_7: \text{count_sr}_{AX}(\text{agent} \rightarrow ?a, \text{truster} \rightarrow ?s, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id) :- \\ \text{count_rating}(\text{rating} \rightarrow ?id, \text{truster} \rightarrow ?s, \text{trustee} \rightarrow ?x), \\ \text{not}(\text{known}(\text{agent}_1 \rightarrow ?a, \text{agent}_2 \rightarrow ?s)).$$

Rule r_4 determines which agents are considered as known, whereas the rest of the rules, r_5 to r_7 , categorize the counted ratings in PR_{AX} (direct experience), KR_{AX} («known» witness) and SR_{AX} (strangers' witness), respectively. In r_7 , we use *negation as failure*, which means that if *known()* fails during execution then *not(known())* will succeed, in order to determine which agents are considered totally strangers. Notice that the above rules are strict ones, i.e. their conclusions cannot be disputed.

The final decision making process for the PR_{AX} is based on a relationship theory among the rating categories. In Fig. 1, we presented the complete relationship among all rating categories, whereas, below, we present three potential theories based on that relationship. In the first theory, all categories count equally, hence, if ratings from all

of them are available (r_8 to r_{10}), then they will all participate in the final reputation estimation. To this end, if one of them is missing, then the other two are combined, whereas if just one category is available, then just that will be taken into account.

$$\begin{aligned}
r_8: & \text{participate}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{AX}) := \\
& \text{count_pr}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{AX}). \\
r_9: & \text{participate}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{KX}) := \\
& \text{count_kr}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{KX}). \\
r_{10}: & \text{participate}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{SX}) := \\
& \text{count_sr}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x, \text{rating} \rightarrow ?id_rating_{SX}).
\end{aligned}$$

In the rest two theories, opinions from different categories conflict each other (conflicting literals), therefore the conflict is being resolved via adding superiority relationships. Specifically, personal opinion is the most important, and then comes friends' opinion and then strangers'. We will present only the superiority relationships and we will not duplicate the rules. The conflict set (for both theories) is:

$$\begin{aligned}
C[\text{participate}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x)] = \\
\{ \neg \text{participate}(\text{agent} \rightarrow ?a, \text{trustee} \rightarrow ?x) \} \cup \\
\{ \text{participate}(\text{agent} \rightarrow ?a1, \text{trustee} \rightarrow ?x1) \mid ?a \neq ?a1 \wedge ?x \neq ?x1 \}
\end{aligned}$$

In the second theory, the priority relationship among the rules is based on the fact that an agent relies on its own experience if it believes it is sufficient, if not it acquires the opinions of others, much as do humans in real life.

$$r_8 > r_9 > r_{10}$$

In the third theory, on the other hand, if direct experience is available (PR_{AX}), then it is preferred to be combined with ratings from known agents (KR_{AX}), whereas if it is not, HARM acts as a pure witness system.

$$r_8 > r_{10}, r_9 > r_{10}$$

3.3 Temporal Defeasible Logic Extension

Since agents may change their objectives at any time, in dynamic environments such as MASs, evolution over time is important and should be taken into account in a trust model. For instance, a typical dishonest agent could provide quality services over a period to gain a high reputation score, and then, profiting from that high score could provide low quality services. Hence, in the temporal extension of HARM, each rating and each conclusion has time duration. To this end, we represent each rating as a persistent temporal literal and each rule conclusion as an expiring temporal literal of TDL. Hence, the rating records are always there, in the model's authority, but only the latest ratings (latest time offsets) will participate in the estimation, providing up-to-date reputation estimation. The truster's rating (r) is fully expressed below (it is active after $time_offset$ time instances have passed and is valid thereafter):

$$\begin{aligned}
\text{rating}(id \rightarrow value_1, \text{truster} \rightarrow value_2, \text{trustee} \rightarrow value_3, \text{validity} \rightarrow value_4, \\
\text{completeness} \rightarrow value_5, \text{correctness} \rightarrow value_6, \text{response_time} \rightarrow value_7, \\
\text{confidence} \rightarrow value_8, \text{transaction_value} \rightarrow value_9) @ time_offset.
\end{aligned}$$

To this end, the rules presented in the previous section are modified accordingly. Each rating is active after t time instances have passed (“@ t ”) whereas each conclusion has a duration (“:duration”). Additionally, each rule has a delay which models the delay between the body and the head, indicating the delay between the cause and the effect. For instance, rule r_1 , which defines that ratings with high confidence and transaction values will be taken into account in the reputation estimation, is modified accordingly as presented below. In particular, a valid rating ($t < t_{current}$) that has confidence and transaction values greater than or equal to user’s thresholds will be considered available for a specific time period ($count_rating:duration$).

$$r_1: count_rating(rating \rightarrow ?idx, truster \rightarrow ?a, trustee \rightarrow ?x):duration :=^{\text{delay}} \\ confidence_threshold(?conf), transaction_value_threshold(?tran), \\ rating(id \rightarrow ?idx, confidence \rightarrow ?conf_x, transaction_value \rightarrow ?tran_x) @t, \\ ?conf_x \geq ?conf, ?tran_x \geq ?tran.$$

The rest of the rules, r_4 to r_{10} , are also modeled in TDL. For instance, rule r_5 , that distinguishes ratings reported by the agent itself (direct experience), is presented below. In other words, a valid rating ($t < t_{current}$), that fulfills the participation restrictions ($count_rating:duration$) at a specific time period ($duration$), is recognized as valid direct experience ($count_pr_{AX}:duration$) in that period if so.

$$r_5: count_pr_{AX}(agent \rightarrow ?a, truster \rightarrow ?a, trustee \rightarrow ?x, rating \rightarrow ?id):duration :=^{\text{delay}} \\ count_rating(rating \rightarrow ?id, truster \rightarrow ?a, trustee \rightarrow ?x):duration, \\ rating(rating \rightarrow ?id, truster \rightarrow ?a, trustee \rightarrow ?x)@t.$$

4 Evaluation

In order to use and evaluate HARM, we implemented the model in EMERALD [16], a framework for interoperating knowledge-based intelligent agents in the SW. It is built on JADE [2], a reliable and widely used multi-agent framework. The main advantage of EMERALD is that it provides a safe, generic, and reusable framework for modeling and monitoring agent communication and agreements. Among others, EMERALD proposes the use of Reasoners [17]. Reasoners are agents that offer reasoning services to the rest of the agent community. A Reasoner can launch an associated reasoning engine, in order to perform inference and provide results. Currently, the framework supports among others four Reasoners that use defeasible reasoning, among them is the DR-Reasoner (based on DR-Device defeasible logic system [1]), the defeasible reasoner that was used for the evaluation.

Furthermore, since HARM is a centralized reputation model we implemented an agent, called *HARMagent*, which acts as the central authority and is responsible for collecting, storing, and keeping the reports safe and available. However, since, currently, there is no available temporal defeasible logic inference engine to deploy directly our temporal defeasible trust model, we transformed our rule-based model into straightforward defeasible logic rules, using temporal predicates in order to simulate the temporal semantics. For the evaluation purposes, we use a testbed designed in [11] with slight changes, adopted from [12] and previously also used in [18]. Mention that,

we preserved the testbed design and the evaluation settings in order to compare the evaluation conclusions. Below, a description of the testbed is given, and next the methodology and the experimental settings for our experiments are also presented.

The testbed environment for evaluating HARM is a multi-agent system consisting of agents providing services and agents that use these services. We assume that the performance of a provider (and effectively its trustworthiness) is independent from the service that is provided. In order to reduce the complexity of the testbed's environment, it is assumed that there is only one type of service in the testbed and, as a result, all the providers offer the same service. Nevertheless, the performance of the providers, such as the quality of the service, differs and determines the utility that a consumer gains from each interaction (called $UG \equiv$ utility gain).

Each agent interaction is a simulation round. Events that take place in the same round are considered simultaneous. The round number is used as the time value for events. In HARM, if a consumer agent needs to use the service, it can contact the centralized authority in order to be informed about the reputations of the provider agents. The consumer agent will select one provider to use its service, the one with the highest value of reputation (fig. 2). The selection process relies on the trust model to decide which provider is likely to be the most reliable. Consumer agents without the ability to choose a trust model will randomly select a provider from the list.

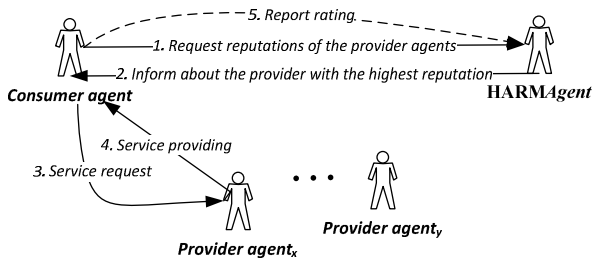


Fig. 2. HARM transaction steps

Firstly, the consumer agent selects a provider, then, it uses the service of the selected provider and gains some utility from the interaction (UG). The value of UG varies from -10 to 10 and it depends on the level of performance of the provider in that interaction. A provider agent can serve many users at a time. After an interaction, the consumer agent rates the service of the provider based on the level of performance and the quality of the service it received. It is assumed that all agents exchange their information honestly in this testbed. This means an agent (as a witness) provides its true ratings as they are without any modification.

The testbed in each experiment is populated with provider and consumer agents. Each consumer agent uses a particular trust model, which helps it selecting a provider when it needs to use a service. In this evaluation, we used HARM, T-REX [18], SETM [15], and NONE (no trust mechanism). The only difference among consumer agents is the trust models that they use, so the utility gained by each agent through

simulations will reflect the performance of its trust model in selecting reliable providers. Hence, the testbed records the UG of each interaction with each trust model used. In order to obtain an accurate result for performance comparisons between trust models, each one will be employed by a large number of consumer agents.

Table 1. Experimental Setting

Number of simulation: 500 / Number of providers: 100	
Good providers	10
Ordinary providers	40
Intermittent providers	5
Bad providers	45

After 500 simulations and 100 participants, as in [12][18], (Table 1 displays the setting), we figured out that the performances of T-REX, as expected, is essentially the same as in [18]. The performances of SERM, which is based just on the weighted sum of the relevant ratings is considerably low, whereas the performance of the NONE group, formed by selecting providers randomly without any trust evaluation, is, also as expected, consistently the lowest. The rest of the groups, based on centralized mechanisms, were able to gather ratings about all interactions in the system. This allows agents using them to achieve higher performance right from the first interactions (Table 2 displays the average UG per interaction).

Table 2. Average UG per interaction

HARM	5.73
T-REX	5.57
SETM	2.41
NONE	0.16
CR	5.48
SPORAS	4.65

Furthermore, experiments with the same setting, presented in [12], have shown that the SPORAS [26] and Certified Reputation (CR) [12] are beneficial to consumer agents, helping them obtain significantly high UG. This means that the tested trust models can learn about the provider's population and allow their agents to select profitable providers for interactions. In contrast, since each provider only shows a small number of ratings to agents using CR, they spend the first few interactions learning about their environment. The slightly higher performance of HARM, is because it is a centralized model (like T-REX) and, thus, able to gather much more information than decentralized models additionally to using a dynamic (defeasible) reputation estimation mechanisms. Furthermore, HARM could not be used to its full potential due to lack of a temporal defeasible inference engine. This is, also, the reason for the almost similar performance between T-REX and HARM.

5 Related Work

Trust and reputation are key ingredients to most multi-agent systems and as a result many different metrics have already been proposed. SPORAS [26] is one of the most notable of these models. In this model, each agent rates its partner after an interaction and reports its ratings to the centralized repository. The received ratings are then used to update the global reputation values of the rated agents. In SPORAS, new agents start with a minimum value; later, reputation values are updated according to the feedback provided by other agents that are involved. Both in HARM and SPORAS ratings are discounted over time, so that the most recent ratings weigh more in an agent's reputation evaluation. Moreover, both models use a learning formula for the updating process so that the reputation value can closely reflect an agent's performance, at any time. However, SPORAS has limitations and, as a result, it is not as dynamic as HARM that uses nonmonotonic logic, much like humans' logic. Furthermore, in SPORAS newcomers are not supposed to be reliable and the other agents do not trust them easily. On the other hand, our approach overcame the problem by evaluating new agents with a neutral rating value.

Certified Reputation [12] is a decentralized reputation model involving each agent keeping a set of references given to it from other agents. In this model, each agent is asked to give certified ratings of its performance after every transaction. The agent then chooses the highest ratings and stores them as references. Any other agent can then ask for the stored references and calculate the agent's certified reputation. This model overcomes the problem of initial reliability in a similar way with HARM. However, opposed to our approach, this model is designed to determine the access rights of agents, rather than to determine their expected performance. Furthermore, it is a witness-based model, whereas HARM combines both witnesses and direct experience, providing a rule-based methodology to deal with the discrimination issue.

T-REX [18] is a centralized reputation model presented by the authors of this paper. It is a hybrid agent trust model based on both witness reputation and personal experience, similar to HARM. However, HARM uses a knowledge-based time-related defeasible theory (using temporal defeasible rules) for the decision making process, whereas T-REX depends on a static user's opinion (provided weights at startup). Additionally, they both deal with the time issue using a similar philosophy, but with a totally different approach. T-REX is based on a monotonic approach, where all past ratings are taken into account, although they lose their importance through a linear extinguishing function over time. In T-REX, a time stamp is used to organize the ratings: ratings with a higher time stamp are considered more important as they refer to more recent evaluations. On the other hand, HARM is based on a rule-based non-monotonic approach, where ratings are considered more like expiring records. In HARM, each rating is characterized by a time offset property, which indicates the time instances that should pass in order to consider the rating active. Additionally, each rating counts only for a time duration, since the rules used in the decision making process characterize them valid only for a specific amount of time. Hence, HARM takes into account only ratings that fulfill the above restrictions, which reflect a more realistic opinion similar to human logic. Comparing, these two models, we believe

that HARM improves T-REX's approach by providing a novel knowledge-based mechanism that improves effectiveness and intuitiveness and that is more related to the traditional human reasoning for assessing trust in the physical world.

6 Conclusions and Future Work

This paper presented HARM, a hybrid, rule-based reputation model which uses *temporal defeasible logic* in order to combine *interaction trust* and *witness reputation*. HARM provides a stable and reliable estimation mechanism, under a centralized administration authority, overcoming the difficulty to locate witness reports. It is based on well-established estimation parameters [3][4], such as information correctness, completeness, and validity, as well as agent's response time. One of the main advantages of HARM is its ability to simulate human thinking and decision making. It is the first model that uses knowledge, in the form of defeasible logic, explicitly in order to predict agent's future behavior. Furthermore, its mechanism can be adopted in any multi-agent system in the Semantic Web, such as JADE. This paper also provided an evaluation of the model's performance and a comparison with four other models.

As for future directions, first of all, we plan to analyze and integrate TDLParser, a temporal defeasible reasoning engine presented in [25], in EMERALD, or extend the DR-Device defeasible logic system [1] to handle temporal modalities. Next, we plan to evaluate our model using TDLParser or the enriched DR-Device system and compare our model's performance with other centralized and decentralized models from the literature, in order to estimate its performance more accurately. Finally, it would be interesting to combine HARM and T-REX or even develop a distributed version of HARM and verify its performance in real-world e-commerce applications, combining it also with Semantic Web metadata for trust [5][6].

References

1. Bassiliades, N., Antoniou, G., Vlahavas, I.: A Defeasible Logic Reasoner for the Semantic Web. *Int. J. on Semantic Web and Information Systems* 2(1), 1–41 (2006)
2. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A white Paper. *EXP in Search of Innovation* 3(3), 6–19 (2003)
3. Castelfranchi, C., Falcone, R.: *Trust Theory: A Socio-Cognitive and Computational Model*, 1st edn. Wiley Series in Agent Technology (2010) ISBN-13: 978-0470028759
4. Castelfranchi, C., Tan, Y.-H.: *Trust and Deception in Virtual Societies*, 1st edn. Springer (2001) ISBN-13: 978-0792369196
5. Ceravolo, P., Damiani, E., Viviani, M.: Adding a Trust Layer to Semantic Web Meta-data. In: Ceravolo, P., et al. (eds.) *Adding a Trust Layer to Semantic Web Metadata*. STUDEFUZZ, vol. 197, pp. 87–104. Springer, Heidelberg (2006)
6. Ceravolo, P., Damiani, E., Viviani, M.: Bottom-Up Extraction and Trust-Based Refinement of Ontology Metadata. *IEEE Transactions on Knowledge and Data Engineering* 19(2), 149–163 (2007)

7. Governatori, G., Rotolo, A.: BIO Logical Agents: Norms, Beliefs, Intentions in Defeasible Logic. *Journal of Autonomous Agents and Multi-Agent Systems* 17(1), 36–69 (2008)
8. Governatori, G., Rotolo, A.: On the Complexity of Temporal Defeasible Logic. In: 13 International Workshop on Non-Monotonic Reasoning, NMR 2010 (2010)
9. Governatori, G., Terenziani, P.: Temporal Extensions to Defeasible Logic. In: Orgun, M.A., Thornton, J. (eds.) *AI 2007. LNCS (LNAI)*, vol. 4830, pp. 476–485. Springer, Heidelberg (2007)
10. Hendler, J.: Agents and the Semantic Web. *IEEE Intelligent Systems* 16(2), 30–37 (2001)
11. Huynh, D., Jennings, N.R., Shadbolt, N.R.: An integrated trust and reputation model for open multi-agent systems. *Journal of AAMAS* (2006)
12. Huynh, D., Jennings, N.R., Shadbolt, N.R.: Certified reputation: How an agent can trust a stranger. In: *AAMAS 2006: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, Hokkaido, Japan (2006)
13. Kontopoulos, E., Bassiliades, N., Antoniou, G.: Visualizing Semantic Web proofs of defeasible logic in the DR-DEVICE system. *Knowledge-Based Systems* 24(3), 406–419 (2011)
14. Koons, R.: Defeasible Reasoning. *Stanford Encyclopedia of Philosophy*. Stanford University (2009), <http://plato.stanford.edu/entries/reasoning-defeasible/>
15. Kravari, K., Bassiliades, N.: Advanced Agent Discovery Services. In: *International Conference on Web Intelligence, Mining and Semantics*, Craiova, June 13-15 (2012)
16. Kravari, K., Kontopoulos, E., Bassiliades, N.: EMERALD: A Multi-Agent System for Knowledge-Based Reasoning Interoperability in the Semantic Web. In: Konstantopoulos, S., Perantonis, S., Karkaletsis, V., Spyropoulos, C.D., Vouros, G. (eds.) *SETN 2010. LNCS*, vol. 6040, pp. 173–182. Springer, Heidelberg (2010)
17. Kravari, K., Kontopoulos, E., Bassiliades, N.: Trusted Reasoning Services for Semantic Web Agents. *Informatica: International Journal of Computing and Informatics* 34(4), 429–440 (2010)
18. Kravari, K., Malliarakis, C., Bassiliades, N.: T-REX: A Hybrid Agent Trust Model Based on Witness Reputation and Personal Experience. In: Buccafurri, F., Semeraro, G. (eds.) *EC-Web 2010. LNBIP*, vol. 61, pp. 107–118. Springer, Heidelberg (2010)
19. Maher, M.J.: Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* 1(6), 691–711 (2001)
20. Nute, D.: Defeasible Reasoning. In: *20th Int. C. on Systems Science*, pp. 470–477. IEEE (1987)
21. Pollock, J.L.: How to reason defeasibly. *Artificial Intelligence* 57 (1992)
22. Pollock, J.L.: Perceiving and Reasoning about a Changing World. *Computational Intelligence* 14, 498–562 (1998)
23. Ramchurn, S.D., Huynh, D., Jennings, N.R.: Trust in multi-agent systems. *Knowledge Engineering Review* 19(1), 1–25 (2004)
24. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. *Communications of the ACM* 43(12), 45–48 (2000)
25. Rubino, R., Rotolo, A.: A Java Implementation of Temporal Defeasible Logic. In: Governatori, G., Hall, J., Paschke, A. (eds.) *RuleML 2009. LNCS*, vol. 5858, pp. 297–304. Springer, Heidelberg (2009)
26. Zacharia, G., Maes, P.: Trust management through reputation mechanisms. *Applied Artificial Intelligence* 14(9), 881–908 (2000)

Whispering Interactions to the End User Using Rules

Benjamin Jailly¹, Christophe Gravier¹, Julien Subercaze¹,
Marius Preda², and Jacques Fayolle¹

¹ Université de Lyon, F-42023, Saint-Étienne, France,
Université de Saint-Étienne, Jean Monnet, F-42000, Saint-Étienne, France,
Télécom Saint-Étienne, école associée de l'Institut Télécom,
F-42000, Saint-Étienne, France,
Laboratoire Télécom Claude Chappe (LT2C), F-42000, Saint-Étienne, France

² ARTEMIS, Télécom SudParis, Institut Télécom
{benjamin.jailly, christophe.gravier,
julien.subercaze, jacques.fayolle}@telecom-st-etienne.fr,
{maris.preda}@it-sudparis.eu

Abstract. The issue of e-Maintenance, i.e. the remote maintenance of devices, is more and more important as the loose-coupling between center of competencies and productions sites is increasing. Maintenance operators have to know procedures for all devices they are in charge of. Whispering possible interactions to operators when they operate could ease their task and increase their productivity. In this paper, we propose to model maintenance procedures (therefore sequences of actions of human operators) using the Semantic Web standards. The system can infer the possible next actions from the human operator, and assist him by suggesting the next operation steps. An analysis of the use's trace is also proposed in order to hint operators on improvements in his processes.

Keywords: Semantic Web, Sequences, Rules, Human Computer Interface.

1 Introduction

1.1 Computer-Aided Maintenance

Online engineering refers to the ability to perform remote operations on real devices over the Internet. In the field of Computer-Aided Maintenance (henceforth CAM), online engineering help industries to execute maintenance procedures on remote devices. Production and industrial maintenance sites rely on highly qualified technicians and staff. The latter are responsible for many assembly/disassembly operations, calibration, etc. These human operators are expected to understand and memorize a large panel of procedure steps, in a specific order. Moreover, moving to the production site presents several issues (cost, risk, etc.). As a consequence, a first maintenance operation is usually performed at distance.

Learning and being able to reproduce the full catalog of maintenance operations can be a difficult task. The complexity of this task is also rapidly exploding since the number of devices operated by a single human operator increases when these devices can be operated remotely. CAM systems must provide a formalism to express such procedures in order to ease operators actions and increase their productivity. In our point of view, on the Human Computer Interface (HCI) side, the CAM systems must also provide indications on which action has been made and "whisper" to the operator the next possible actions in the maintenance procedures. The aim is to limit the number of human errors, as well as increasing the efficiency (time spent, quality, etc.) of the maintenance process.

1.2 Use Case Scenario

In order to illustrate our proposition, we put forward the following scenario. Alice is a technician working in a manufacture. Her device presents several malfunctions. She calls Bob who is an operator in a company performing maintenance services. In order to decrease the maintenance service cost, Bob first runs remote operations on the device to avoid *in situ* maintenance, if possible. But since Bob conducts maintenance on several devices for different companies, he does not know by heart all the procedures for all the machines. He uses the CAM system provided by his company to run the supervision that includes all the procedures for the many devices he is in charge of.

This scenario implies several issues.

- *Procedure formalism.* The CAM system must provide a formalism to describe procedures.
- *Business Logic.* The business logic determines which procedure is being performed and what are the next possible actions. This logic must be loosely-coupled with any programming language in order to be fully reusable.
- *Analysis.* The CAM system should provide tools to analyze actions that have been made by the operators, in order to propose improvements..

In this paper, we propose a framework to express maintenance procedures based on Semantic Web technologies. A rule-based engine provides the context logic to indicate the operator which next possible actions can be made according to the current procedures. The paper is organized as follows. Next section presents the related work. Section 3 introduces the general architecture of our solution. Section 4 details the implementation of our proposition. Section 5 concludes and indicates future works.

2 Related Work

In the literature, many e-Maintenance systems are built on top of Semantic Web technologies [1]. They provide a high level abstraction and description of resources. Coupled with rules engines, they offer reasoning capacities. In [2], the authors propose a complete conceptual framework for e-Maintenance. This

framework helps designing e-Maintenance applications using five abstractions levels such as strategic vision, business processes, organization, service, data structure, and IT infrastructure. The sequences are designed as workflows. The framework does not describe the formalism of the sequences and the way to help operators in maintenance operation.

In [3], the authors propose a user model based on ontologies. This model is used to characterize users in order to adapt Web application using rules. They designed and implemented a rule-driven agent, which tries to capture interesting patterns of user-system interactions. This approach is user-centred but not on processes of pre-defined actions.

In [4], the authors want to provide a framework for personalized e-learning system. The goal is to adapt the system depending on user context. This framework is using ontologies and rules.

In [5], the authors propose a system to monitor and interpret sequential activities in e-learning environments. Their system is based on pattern mining and data clustering. The goal is to offer a personalization of e-learning services based on the results of the analysis. Sequences are represented using Discrete Markov Models. This approach is interesting for e-learning processes since the system is able to determine if users have successfully completed the problem.

3 General Architecture

Rules engines enable a loose-coupling between business rules and programming components. It entails a high level of portability and reuse.

By sequences we imply an ordered list of actions on a HCI that compose a procedure. The usage of Semantic Web technologies to describe sequences of actions for e-Maintenance is motivated by the interoperability they provide, the detachment on specific programming language and mainly the reasoning they provide through rule languages.

In this section, we explain the general architecture of our proposition. It is illustrated in Figure 1 in a form of a middleware. The procedure expert edits the sequences of actions that are serialized in the Knowledge Base (KB). The operator sends actions to the middleware through the HCI's widgets. The Processing Blocks (PB) interprets the actions. It requests the KB through SPARQL Protocol and RDF Query Language (SPARQL) queries, which are the current possible sequences based on previous inferences. PB also put the new action to be the current one in the KB. The Reasoner Bloc (RB) infers which are the next possible interactions according to sequences rules and the Notification Block (NB) pushes the results to the HCI.

We take the following example:

Let $\{A_i\}$ be interactions made on a HCI through widgets. Let $\{S_j\}$ be sequences of interactions pre-determined and serialized in the system. S_j is an ordered list of actions A_i .

Let's assume that :

$$S_1 = \{A_1, A_2, A_3\} \text{ and } S_2 = \{A_1, A_2, A_4\}. \quad (1)$$

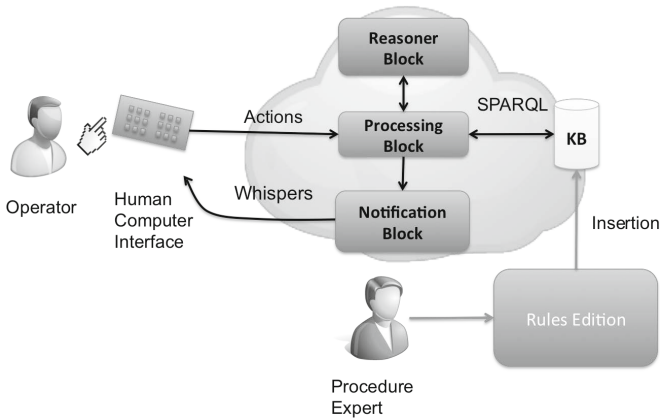


Fig. 1. General architecture

The operator sends the action A_1 . The system whispers that the next possible action recorded is A_2 in sequences S_1 and S_2 . The widget corresponding to A_2 is highlighted in the HCI. The operator then sends A_2 . In S_1 and S_2 , the next actions are respectively A_3 and A_4 , and the corresponding widgets are highlighted in the HCI.

Back to our use case, Bob has now finished to perform maintenance operations on Alice's device. Let's T be the trace of Bob's actions. An example of T could be :

$$T = \{A_3, S_3, S_5, A_3, A_3, S_1, \dots\} \quad (2)$$

where A_i is an action on the CAM system's HCI and S_j is a sequence of actions recorded in the system. Let's assume that in the Business rules of his company:

$$S_5 \equiv S_3 \text{ and } Card(S_5) \gg Card(S_3) \quad (3)$$

The system should notify Bob that he performed one or several useless and/or suboptimal actions in the maintenance procedures. In order to realize it, the full use's trace of Bob has to be serialized in the system before being analyzed.

Since the system could be used by somebody else than Bob, the use's trace need to contain the name of the operator. The system could also be collaborative, i.e. Bob could be using the system in the meantime with James that has other competencies for the task. The analysis of the trace can thus give details on the full remote operation but also details on a single operation.

4 Implementation

4.1 Actions as RDF Resources

RDFS proposes in its core components `rdfs:Container` such as `rdf:Alt`, `rdf:Bag`, `rdf:Seq`. They are used to describe group of things. Using `rdfs:Container` allows

the granularity of expressing compositions of ordered lists and choices of actions, with the power of Semantic Web tools. We define an action as a RDF resource. This resource has 4 properties: `isCurrent`, `isNext`, `isVerified` and `hasName`. The three first ones are XML Schema boolean datatypes, whereas the third one is a XML Schema `normalizedString` datatype:

- `isCurrent`: defines if an action is the current one (i.e. if this action is the one being applied).
- `isNext`: defines if an action is the following one to another. Its value is determined by the rules engine, as explained in the following section.
- `isVerified`: defines if an action has already been called in the sequence.
- `hasName`: defines the name of the action.

The full sequence is then a composition `rdf:Seq`, which represents the sequences of actions to perform, and `rdf:Alt` that represents a choice between actions (e.g. A_3 or A_4). A literal value is predicated to the `rdf:Seq` in order to identify the device.

As framework, Jena seems a good compromise between its RDF graph framework and its reasoner capacities, based on Datalog and RETE [6]. The RDF graphs are stored in Joseki¹, an HTTP engine that supports SPARQL and SPARQL/Update protocols.

4.2 Reasoning for Interactions Hints

The main goal of the rule engine is to determine the possible actions following the current one and depending on the previous ones. Rules can be described as the following pseudo-code inside a sequence:

```
PREFIX:
xsd: http://www.w3.org/2001/XMLSchema#
widg: http://ocelot.ow2.org/2012/Sequence/Actions/Widget#

if {item_{m}, widg:isCurrent, 'true'^^xsd:boolean}
and if {item_{m-n}, widg:isVerified, 'true'^^xsd:boolean}
then
    {item_{m+1}, widg:isNext, 'true'^^xsd:boolean}
```

if the item (m+1) in the sequence has `rdf:Alt` type, we need to put all its elements at next also. In order to verify the triple described above, we implemented custom built-ins in the Jena rule-engine. This built-in is in the form `hasNext(?seq, ?x, ?y)`. It verifies that `{seq rdf:_(n) x}` and `{seq rdf:_(n+1) y}` and that the previous actions in the sequences have been validated.

Following the Open World Assumption, the reasoning system, based on Jena rules engine, may accept new facts that might contradict previous ones. KB revision could be a solution in order to keep consistency. Some works have been conducted in this field such as in [8] and [9]. In [8], the authors introduce an

¹ <http://www.joseki.org/>

annotation approach for prioritizing rules over the remaining ones, in order to prevent contradiction clashes. The authors of [9] submit a system that maintains the internal state of DL reasoners. But this topic is beyond the scope of this article. In order to revise the KB and keep consistency, we therefore decided to remove old assertions and replace them with new deduced ones when a clash occurs.

Once the following actions are determined, the system pulls the "whispers" to the client application using the WebSocket protocol².

4.3 Use's Trace

We also decided to record traces using rdf:Seq for the same above-mentioned reasons. Each recorded action is identified by its name and by the person who made that action.

The use's trace can thus easily be analyzed using pattern matching algorithm such as SuffixTree [10] in order to determine which actions and/or sequences have been made, and by whom.

The RDF resources representing the sequences need a property to express equivalency in order to propose improvements to the operator. A disadvantage using RDF against OWL is that RDF does not provide support for symmetric properties. Whence, when designing sequences, we need to explicitly express that $S_i \Rightarrow S_j$ and $S_j \Rightarrow S_i$. If the system determines that S_5 appears in the use's trace of Bob, the system can notify Bob that he should be considering using S_3 instead. Again, the rules engine is responsible for doing those assumptions. The analysis of use's trace is available in our system through a RESTful API.

5 Conclusion and Future Works

The presented work is part of a global open source project: the Open Collaborative Environment for the Leverage of Online instrumentation, OCELOT³. The goal of the OCELOT framework is to give a high level of interoperability and reuse in the building of online laboratories. It also tries to bring collaboration and group awareness using communication tools and notifications mechanism [11]. This project is hosted by the OW2 forge and is available under the LGPL license.

The "whispering" system presented is integrated in the whole OCELOT framework as a middleware component. This component predicts and hints users on their interactions on HCIs based on pre-defined sequences of actions. Those sequences are modeled using RDF resources and mainly rdf:Seq elements. A rules-based engine based on Datalog and RETE implemented in the Jena framework determines following interactions. A notification block based on WebSockets pushes the determined interactions to the client application that is responsible for giving feedbacks to the operator. The system we propose is also capable of

² <http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76>

³ <http://ocelot.ow2.org>.

making basic analysis on the use's trace and recommending improvements to the operator.

Future works will consist in building a graphical Web interface to build such sequences for non-programmers experts and in improving afterward analysis of use's trace. Afterwards, users tests will be conducted.

References

1. Muller, A., Marquez, A.C., Iung, B.: On the concept of e-maintenance: Review and current research. *Reliability Engineering & System Safety* 93(8), 1165–1187 (2008)
2. Lung, B., Levrat, E., Marquez, A.C., Erbe, H.: Conceptual framework for e-Maintenance: Illustration by e-Maintenance technologies and platforms. *Annual Reviews in Control* 33(2), 220–229 (2009)
3. Lindstaedt, S.N., Beham, G., Kump, B., Ley, T.: Getting to Know Your User – Unobtrusive User Model Maintenance within Work-Integrated Learning Environments. In: Cress, U., Dimitrova, V., Specht, M. (eds.) *EC-TEL 2009*. LNCS, vol. 5794, pp. 73–87. Springer, Heidelberg (2009)
4. Henze, N., Dolog, P., Nejdl, W.: Reasoning and ontologies for personalized e-learning in the semantic web. *Educational Technology & Society* 7(4), 82–97 (2004)
5. Köck, M., Paramythis, A.: Activity sequence modelling and dynamic clustering for personalized e-learning. *User Modeling and User-Adapted Interaction* 21, 51–97 (2011)
6. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19(1), 17–37 (1982)
7. O'Connor, M.F., Knublauch, H., Tu, S., Grosz, B.N., Dean, M., Grosso, W., Musen, M.A.: Supporting Rule System Interoperability on the Semantic Web with SWRL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) *ISWC 2005*. LNCS, vol. 3729, pp. 974–986. Springer, Heidelberg (2005)
8. Grosz, B., Dean, M., Kife, M.: The Silk System: Scalable higher-order defeasible rules. In: *RuleML 2009*. LNCS, vol. 5858, Springer, Heidelberg (2009)
9. Halashek-Wiener, C., Parsia, B., Sirin, E.: Description Logic Reasoning with Syntactic Updates. In: Meersman, R., Tari, Z. (eds.) *OTM 2006*. LNCS, vol. 4275, pp. 722–737. Springer, Heidelberg (2006)
10. Ukkonen, E.: Constructing suffix trees on-line in linear time. In: *Proc. Information Processing*, pp. 484–492. Elsevier (1992)
11. Jailly, B., Gravier, C., Preda, M., Fayolle, J.: Interactive mixed reality for collaborative remote laboratories. In: *Proceedings of the Third International ACM Workshop on Multimedia Technologies for Distance Learning, MTDL 2011*, pp. 1–6 (2011)

Personalizing Location Information through Rule-Based Policies

Iosif Viktoratos¹, Athanasios Tsadiras¹, and Nick Bassiliades²

¹ Department of Economics,
Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece

² Department of Informatics,
Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece

{viktorat, tsadiras, nbassili}@auth.gr

Abstract. In this paper, the idea of providing personalized, location-based information services via rule-based policies is demonstrated. After a short introduction about related technologies and approaches, an innovative Personalized Location Information System (PLIS) is designed and implemented. PLIS delivers personalized and contextualized information to users according to rule-based policies. More specifically, many categories of points of interest (e.g. shops, restaurants) have rule-based policies to expose and deploy their marketing strategy on special offers, discounts, etc. PLIS evaluates these rules on-the-fly and delivers personalized information according to the user's context and the corresponding rules fired within this context. After discussing the design and the implementation of PLIS, illustrative examples of PLIS functionality are presented. As a result, PLIS proves that combining contextual data and rules can lead to powerful personalized information services.

Keywords: RuleML, Rules, Location Based Services, Context, Points of Interest, Jess.

1 Introduction

1.1 Rules and Policies

Rule-based policies are an important sector of our everyday life. They are used consistently by various types of businesses (or in general, Points of Interest-POI in our Location-based context), not only to deploy their marketing strategy, but also to expose them to the public in a comprehensible manner. A constraint is that, such kind of policies, have to be translated into a computer understandable language, in order to be executed and adopted by an information service [1]. As a result, a general rule language is needed for this purpose.

After various initial efforts in conventional languages [1], RIF was adopted as a general rule language [2] by the World Wide Web Consortium (W3C). RIF was influenced by a previous but still ongoing rule standardization initiative called RuleML [3]. RuleML is a family of sublanguages which are used to publish rules on

the web [4] and their main objective is to provide a useful rule markup approach for semantic web applications [5]. RuleML was widely accepted by scientific community and it was chosen as a general technology for various reasons. First of all, it is a powerful markup language (XML with a predefined Schema) which supports various types of rules such as deductive, reactive and normative [1]. As an XML-based language, RuleML addresses the issues of interoperability and flexibility among different systems on the web, by allowing rules to be encoded in a standard way [1, 6]. Last but not least, beyond representation, rules need to be translated to an inference engine (such as Jess, Drools, Prova, etc.) in order to be executed by a machine [7-9].

1.2 Location-Based Services and Related Work

On the other hand, the technological revolution on Smartphone's capabilities and related technologies, such as semantics, made Location Based Services (LBS) very popular and led to a huge growth [10-13]. They are used daily by millions of people for e.g. for navigation, for tracking, even in emergency situations [10].

LBS user's environment changes continuously, so it is really important for successful LBS to deliver up-to-date personalized and contextualized information to user [14-18]. Researchers and industries are working in various sectors to evolve such services. Approaches related to the service proposed in this paper are described below.

Latest LBS combine semantics (ontologies, rules) with smartphone's capabilities (GPS, sensors) to deliver contextualized information [19-21]. Many approaches also use social media data for personalized POI recommendations [22,23]. Another interesting sub-sector of LBS is mobile search optimization. Up to date LBS offer high quality mobile search capabilities by personalizing query results or search tag recommendations [24-26].

1.3 Motivation-Overview

The aim of the work presented to this paper is to combine semantics with location information services to deliver personalized and contextualized information to users. A system called "PLIS - Personalized Location Information System" was implemented for this purpose.

Our proposed system uses semantics due to the fact that a) these technologies are improving knowledge sharing and interoperability between systems and b) these technologies are well-suited for the representation of various policies, that is suitable for our case. Moreover, a rule-based approach was followed for PLIS implementation, so as to enable higher quality context perception. Rule-based systems are capable of understanding context changes and responding accordingly with intelligence to different situations without user intervention. Such systems are more autonomous [27, 28]. Concerning the adoption of RuleML, except from the general advantages referred previously (interoperability, flexibility, rules representation), it fits perfectly to our case for another reason. Because of the fact that PLIS users are capable of adding rules at run-time, an xml-based user friendly language is desirable.

PLIS could be easily combined with most of the existing approaches. On the other hand it is different because it enables a dynamic rule base by offering users the option to add rules at run time. Next section demonstrates the general idea, the design

and the implementation of the system. In section 3, PLIS's functionalities become clear by the use of an illustrative example. Finally, section 4 concludes the paper and discusses future directions.

2 Design and Implementation

PLIS provides a general interface for connection between POI owners and potential customers-users. In everyday life, many points of interest are businesses (for example restaurants), and as businesses, they have a rule-based policy in order to deploy their specific business or marketing strategy (for example a restaurant offering a free meal to children under 10 years old on Fridays). The general idea is to combine POI'S rule-based policies with user's context to deliver personalized, up-to-date information. Every time a user logs into the system to search for a point of interest, PLIS gets user's context, evaluates the rules associated with nearby POIs and delivers personalized information to user, depending on the rules fired. The personalized information is presented to the user on the well known Google Maps (<http://maps.google.com>). The general design of PLIS is illustrated in Figure 1.

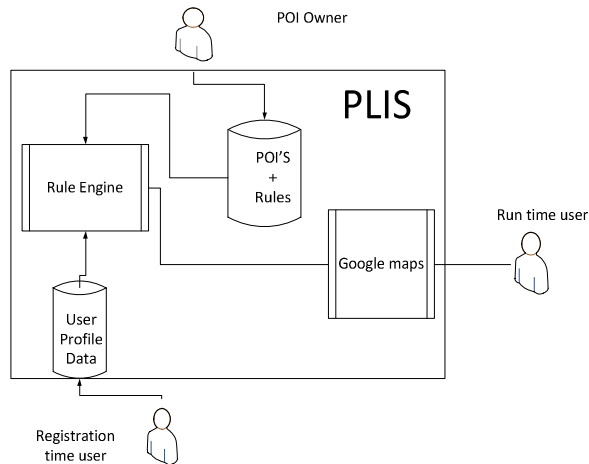


Fig. 1. PLIS design

Various types of rules related to user context are supported by the system. PLIS vision is to be able to implement every possible rule, given that there is flexibility in the schema of the data kept by the system or even linked data found in the web. PLIS currently is able to handle rules concerning user's occupation (e.g. a restaurant offers discount to students), gender, age, location (e.g. a coffee shop decreases prices for users who are less than 200 meters away) and time. To be more specific, PLIS possesses the following functionalities (see figure 2, each operation has a number indicating the corresponding step):

A: User's Registration:

— A1. User registers to the system by completing a registration form so as PLIS to build a profile (registration time user).

– A2. User profile data such as first name, last name, occupation, gender, age, city, state, e.t.c are stored in the database.

B: Insertion of Points of Interest

– B1. User is able to insert his own POI’s (via JSP server) accompanied by their own rule based policy. Because this policy can change e.g. by antagonism, the authorized editing of the corresponding rule base is allowed by the POI’s owners.

– B2. All the data concerning the POI accompanied by its rules, are saved to the corresponding database.

C: Presentation of Personalized information

To present the personalized information to the end user, the following steps are made:

• **Step C1:**

- a. After registration, user is able to log into the system.
- b. System checks user profile database for authentication.

• **Step C2:** JSP collects user context (profile, location, time,day). (run time user).

• **Step C3:**

- a. For every POI, rules (if any) are being fetched (by JSP), along with relevant attribute values (for example price, etc).
- b. Rules (after being transformed as shown above), POI data and user context attribute values are asserted to the Jess rule engine.

• **Step C4:** Jess rule engine evaluates rules using the asserted facts and updates POIs’ attribute values according to the rules fired depending on user’s context. The new values are fetched by JSP.

• **Step C5:** Finally, data transfer to client is performed for visualization and personalized information provision.

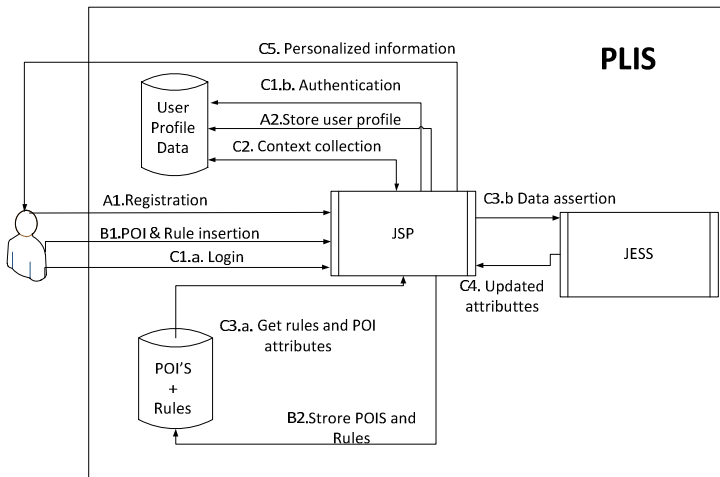


Fig. 2. System operation process

To implement PLIS, the use of Reaction RuleML for rules representation is a clear choice based on the discussion in section 1. This subcategory of RuleML was chosen because such kinds of policies are usually represented by production rules and Reaction RuleML is up to that task [29]. Jess was chosen as an inference engine because of the fact that it is a lightweight rule engine and connects well with standard web technologies which were needed for PLIS system implementation (JSP, html, Javascript). To merge the two above design choices (RuleML and Jess), a transformation from rules in RuleML format to Jess rules is needed. This was done by developing such a transformation using XSLT technology [30]. For example in figure 3, the rule “if a person is a student, discount price 20% and store the new offer” is represented a) in Reaction RuleML and b) in Jess, with the transformation from a) to b) to be made by the use of our developed XSLT file.

<p>RuleML representation</p> <pre> <RuleML... xsi:schemaLocation="http://www.ruleml.org/0.91/ xsd http://ruleml.org/reaction/0.2/dr.xsd "> <Assert> <Rule style="active"> <label>student_discount</label> <if> <And> <Atom> <Rel>person</Rel> <slot> <Ind>pid</Ind> <Var>x</Var> </slot> <slot> <Ind>occupation</Ind> <Ind>student</Ind> </slot> </Atom> <Atom> <Rel>service</Rel> <slot> <Ind>sid</Ind> <Var>y</Var> </slot> </pre> <p style="text-align: right;">Continued here →</p>	<pre> <slot> <Ind>price</Ind> <Var>p</Var> </slot> </Atom> </And> </if> <then> <Equal> <Ind>offer</Ind> <Expr> <Fun>discount</Fun> <Var>p</Var> <Data>0.2</Data> </Expr> </Equal> </then> </Rule> </Assert> </RuleML> </pre>
<p>Jess equivalent after transformation</p> <pre> (defrule student_discount (person (pid ?x) (occupation student)) (service(sid ?y) (price ?p)) =>(store offer (discount ?p 0.2))) </pre>	

Fig. 3. Rule representation in Reaction RuleML and Jess

3 Demonstration of PLIS

The PLIS capabilities are demonstrated, based on data from various POIs of the city of Thessaloniki, Greece that are also accompanied by rules specifying the marketing policy of the POI. Two different user profiles are used as an example (Table 1).

Table 1. Two different user profiles

Profile					Environment		
	Name	Occupation	Gender	Age	Time	Day	Location
User A	Bob	Student	Male	22	22:45	Thursday	Location A
User B	Mary	Unemployed	Female	35	19:10	Friday	Location B

On the other hand, except from the above profiles, a random place from the database selected for testing (Table 2). Table 2 shows default values for attributes ‘average price per person’ and ‘minimum order’ and also the rules attached to this place.

Table 2. A random place chosen for demonstration

Name of Place	Average price per person (€)	Minimum order (€)	Rule 1	Rule 2
Pasta Pizza	10	5	Decrease minimum order 20% for students which are closer than 200m after 22:00	Discount average price 10% for unemployed women on Fridays

For demonstrating better the capabilities of PLIS, a scenario about User A and B will be presented.

Scenario. As soon as “Bob” (User A) logs into PLIS, rule 1 is fired for place A (because he is a student, assuming his distance from place A is closer than 200m and time is after 22:00 o’clock). Considering this rule, minimum order for Bob is 20% less (4€) for this place. PLIS evaluates rules and delivers personalized information to Bob (Figure 4). Similarly, when user B (Mary) logs into the system, one rule (rule 2) is fired for place A (because Mary is an unemployed woman and current day is Friday). Taking these under consideration, average price per person for Mary at place A is 10% less (9€). The delivered information to Mary is illustrated in figure 5. This scenario illustrates how the delivered information is displayed to the end user and the capabilities of PLIS representing rules concerning a) gender, b) day c) location d) occupation, e) time and f) a non-applicable rule case (rule 1 for user B, rule 2 for user A).

4 Conclusions and Future Work

Embedding rules to location-based information systems can offer a boost to the quality of delivered information. By developing PLIS, the viability of this idea was clearly demonstrated. Moreover, a capability of adding rules on the fly can not only lead to powerful, autonomous and intelligent services, but also to the evolution of these services. Experimental testing, confirmed PLIS evolution (as soon as more rules added to the system) without developers intervention.

PLIS implementation can evolve in various ways. First of all, because of the fact that POI owners are unfamiliar with RuleML, a convenient (probably visual) RuleML editor could be embedded [31]. Furthermore, in our future plans is to use OWL and/or RDF data (as in linked data) to represent user profiles and POI related information, for greater flexibility. Moreover, a mobile application e.g. for a Smartphone, can be implemented and integrated with the native context sensing devices (e.g. GPS).

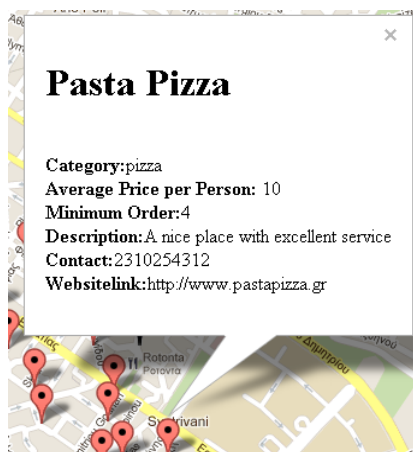


Fig. 4. Information for user “Bob”

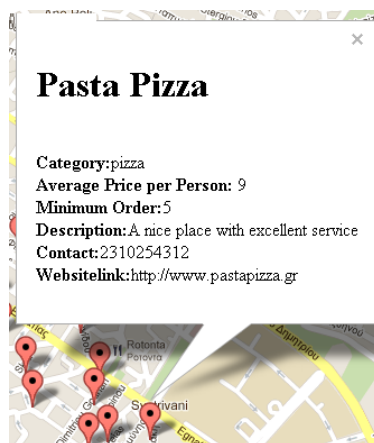


Fig. 5. Information for user “Mary”

References

1. Hu, Y.-J., Yeh, C.-L., Laun, W.: Challenges for Rule Systems on the Web. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 4–16. Springer, Heidelberg (2009)
2. Michael, K., Harold, B. (eds.): RIF Overview, <http://www.w3.org/2005/rules/wiki/Overview>
3. Boley, H., Tabet, S., Wagner, G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In: Proc. SWWS 2001, Stanford (July/August 2001)
4. Li, J.: Rule-Based Social Networking for Expert Finding. A thesis submitted in partial fulfillment of the requirements for the degree of Master of Computer Science In The Graduate Academic Unit of Computer Science (2004)
5. Mei, J., Bontas, P.: Reasoning Paradigms for OWL Ontologies. Techn. Report, Germany (2004)

6. Eberhart, A.: SmartGuide: An intelligent Information System basing on Semantic Web Standards. In: Proc. of the International Conference on Artificial Intelligence (July 2002)
7. Etter, R., Costa, P.D., Broens, T.: A rule-based approach towards context-aware user notification services. In: Proceedings of the 2006 ACS/IEEE International Conference on Pervasive Services (PERSER 2006), Washington, DC, USA, pp. 281–284 (2006)
8. Liang, S., Fodor, P., Wan, H., Kifer, M.: OpenRuleBench: An Analysis of the Performance of Rule Engines. In: WWW 2009, Madrid (2009)
9. Friedman-Hill, E.: Jess in Action. Rule-Based Systems in Java, pp. 32–33. Manning Publications (2003) ISBN-10: 1930110898
10. Steiniger, S., Moritz, N., Alistair, E.: Foundation of Location Based Services. Lecture Notes on LBS (2006)
11. Serrano, D., Hervás, R., Bravo, J.: Telemaco: Context-aware System for Tourism Guiding based on Web 3.0 Technology. In: International Workshop on Contextual Computing and Ambient Intelligence in Tourism (2011)
12. Tryfona, N., Pfoser, D.: Data Semantics in Location-Based Services. In: Spaccapietra, S., Zimányi, E. (eds.) Journal on Data Semantics III. LNCS, vol. 3534, pp. 168–195. Springer, Heidelberg (2005)
13. Ilarri, S., Lllarramendi, A., Mena, E., Sheth, A.: Semantics in Location-Based Services. IEEE Internet Computing 15(6), 10–14 (2011)
14. Liu, Y., Wilde, E.: Personalized Location-Based Services. In: Proc. of the iConference (2011)
15. Woerndl, W., Groh, G.: Utilizing physical and social context to improve recommender systems. In: Proceedings of the IEEE Workshop on Web Personalization and Recommender Systems (WPRS), International Conference on Web Intelligence, WI 2007 (2007)
16. Boudighaghen, O., Tamine, L., Boughanem, M.: Context-Aware User’s Interests for Personalizing Mobile Search. In: 12th IEEE International Conference on Mobile Data Management (2011)
17. Hosseini-Pozveh, M., Nematbakhsh, M., Movahhedinia, N.: A multidimensional approach for context-aware recommendation in mobile commerce. (IJCSIS) Int. Journal of Computer Science and Information Security 3(1) (2009)
18. Van Woensel, W., Casteleyn, S., De Troyer, O.: Applying Semantic Web Technology in a Mobile Setting: The Person Matcher. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 506–509. Springer, Heidelberg (2010)
19. Van Woensel, W., Casteleyn, S., Troyer, O.: A framework for decentralized, context aware mobile applications using semantic web technology. In: Proceedings of the Confederated International Workshops and Posters on the Move to Meaningful Internet Systems, pp. 88–97. Springer (2009)
20. Keßler, C., Raubal, M., Wosniok, C.: Semantic Rules for Context-Aware Geographical Information Retrieval. In: Barnaghi, P., Moessner, K., Presser, M., Meissner, S. (eds.) EuroSSC 2009. LNCS, vol. 5741, pp. 77–92. Springer, Heidelberg (2009)
21. Hwang, H., Shin, S., Kim, K., Lee, S., Kim, C.-S.: Context-aware System Architecture using Personal Information based on Ontology. In: 5th ACIS International Conference on Software Engineering Research, Management & Applications (2007)
22. Savage, N.S., Baranski, M., Chavez, N.E., Höllerer, T.: I’m feeling LoCo: A Location Based Context Aware Recommendation System. In: Proc. 8th International Symposium on Location-Based Services in Vienna, November 21-23 (2011)
23. Patton, E.W., McGuinness, D.L.: The Mobile Wine Agent: Pairing Wine with the Social Semantic Web. In: 2nd Social Data on the Web Workshop (2009)

24. Choi, O., Kim, K., Wang, D., Yeh, H., Hong, M.: Personalized Mobile Information Retrieval System. *International Journal of Advanced Robotic Systems* (2012)
25. Boudighaghen, O., Tamine, L., Boughanem, M.: Personalizing Mobile Web Search for Location Sensitive Queries. In: 12th IEEE Int. Conf. on Mobile Data Management (2011)
26. Arias, M., Cantera, J.M., Vegas, J.: Context-Based Personalization for Mobile Web Search. In: *Very Large Data Bases Conference, Auckland, New Zealand*, pp. 23–28 (2008)
27. Lassila, O.: Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help? In: *Semantic Web and Policy Workshop* (2005)
28. Heeps, S., Dulay, N., Lupu, E., Schaeffer-Filho, A.E., Sloman, M., Strowes, S., Sventek, J.: The autonomic management of ubiquitous systems meets the semantic web. In: *The Second International Workshop on Semantic Web Technology For Ubiquitous and Mobile Applications* (2006)
29. Adrian, P., Alexander, K., Harold, B.: A Homogenous Reaction Rule Language for Complex Event Processing. In: *2nd International Workshop on Event Drive Architecture and Event Processing Systems, EDA-PS 2007, Vienna, Austria* (2007)
30. Sherman, G.: *A Critical Analysis of XSLT Technology for XML Transformation*. Senior Technical Report (2007)
31. Kontopoulos, E., Bassiliades, N., Antoniou, G., Seridou, A.: Visual Modeling of Defeasible Logic Rules with Dr-VisMo. *International Journal on Artificial Intelligence Tools* 17(5), 903–924 (2008)

Using *SOIQ(D)* to Formalize Semantics within a Semantic Decision Table

Yan Tang Demey and Trung-Kien Tran

STARLab, Department of Computer Science, Vrije Universiteit Brussel
{yantang, truntran}@vub.ac.be

Abstract. As an extension to decision tables, Semantic Decision Tables (SDTs) are considered as a powerful tool of modeling processes in various domains. An important motivation of consuming SDTs is to easily validate a decision table during the Validation and Verification (V&V) processes. An SDT contains a set of formal agreements called commitments. They are grounded on a domain ontology and considered as a result from group decision making processes, which involve a community of business stakeholders. A commitment contains a set of constraints, such as uniqueness and mandatory, with which we can analyze a decision table. A vital analysis issue is to detect inconsistency, which can arise within one table or across tables. In this paper, we focus on the formalization of the semantics *within* one SDT using the Description Logic *SOIQ(D)*. By doing so, we can use existing reasoners to detect inconsistency and thus assist decision modelers (and evaluators) to validate a decision table.

Keywords: Semantic Decision Table, Conceptual Modeling, Description Logics.

1 Introduction

An important analysis issue for decision tables is Validation and Verification (V&V [6]), the goal of which is to ensure the quality of the modeled decision rules. Validation is a process of checking whether or not the decision rules are correctly modeled according to certain meta-rules (or models). It has the requirements of building a right decision table model. Verification is a process of confirming that the decision rules are correctly built. It has the requirements of building a decision table right. V&V is a mandatory step towards ensuring the consistency and correctness of a decision table. More specifically speaking, validation is to ensure its consistency; and verification is to ensure the correctness.

A Semantic Decision Table (SDT [19]) is a decision table containing semantically rich meta-information and meta-rules. It has been studied and exploited in the EC FP7 Prolix project [2], EC ITEA DIYSE project [3] and other national projects, where it has shown its usefulness in several real-life applications, such as tuning parameters of an algorithm [18,21] and managing data semantics for smart home [16].

¹ V&V is a general problem for business models, which cover decision tables as illustrated in this paper, and other models like decision trees and Bayesian networks etc.

² <http://www.prolixproject.org/>

³ <http://dyse.org:8080/>

An SDT allows rule modelers, knowledge engineers or evaluators to analyze a decision table using domain semantics. It contains a set of formal agreements called *commitments*, grounded on a domain ontology, and, specified by a community of business stakeholders (domain experts). A commitment specifies how to use a binary fact types defined in the ontology. It can be 1) instantiation of a concept or a binary fact type, 2) a constraint, 3) selecting/grouping binary fact types from one or several contexts, 4) instantiation of a value for a concept if its value range is defined in a constraint, 5) articulation, which is a mapping between a concept and the glosses defined in a glossary, dictionary and thesaurus, 6) interpretation and implementation of role pairs, and 7) alignment of concepts within/across contexts.

The process of modeling commitments is also called “commitment grounding”. During this process, the domain experts (e.g., rule editors and business people) specify hidden rules and meta-rules of this decision table in the commitments, which can be stored in Semantic Decision Rule Markup Language (SDRule-ML, [19]). SDRule-ML is based on First-Order Logic Rule Markup Language (FOL RuleML [4, 21]) and developed as a markup language for ontology-based semantic decision support languages and models.

There are a few existing V&V approaches for decision tables. Shwayder [15] proposes combining decision columns in a decision table in order to reduce redundancies. Pooch [13] illustrates a survey on decomposition and conversion algorithms of translating decision tables in order to check for its redundancy, contradiction and completeness. Vanthienen et al. [22] illustrate using PROLOGA [5] (a decision table tool) to discover the intra-tabular anomaly, which is caused by a cyclic dependence between a condition and an action, and inter-tabular anomaly, which is caused by redundancy, ambivalence and deficiency. Qian et al. [14] use the approach of approximation reduction to managing incomplete and inconsistent decision tables. Incomplete and inconsistent decision tables are reduced into complete and consistent sub tables. Other related work can be found in [7,8,11].

Compared to their work, our approach is focused on using ontological axioms as the meta-rules for validating a decision table. As an ontology is shareable and community-based, the SDT validation process thus supports group activities in a nature way. Decision modelers and rule auditors share their common view through this process. By doing so, misunderstanding is minimized and the cost is consequently reduced.

Inconsistency can arise within one decision table or across tables. In this paper, we focus on the former situation. In our previous papers [17,20], we have studied how ontological constraints can be directly used within one SDT and how RDFs/OWL constraints can be mapped from/to SDRule-ML. Yet the formalization and semantics concerning computational properties for validating an SDT remain unanswered, which becomes the paper motivation and our main contribution. In addition, we need to point out that our effort here is restricted to the process of validation in V&V. Verification is out of the paper scope.

In this paper, we formalize the semantics within an SDT using the Description Logic (DL) language *SOIQ(D)*, which has an advantage of the availability of reasoning algorithms and tools. The paper is organized as follows. Sec.2 is the paper background.

⁴ <http://ruleml.orf/fo1>

⁵ <http://www.econ.kuleuven.be/prologa>

The main contribution of this paper is illustrated in Sec.3. We illustrate the related work and discuss the ideas in Sec.4. In Sec.5, we present the conclusion and our future work.

2 Background: $SOIQ(\mathbf{D})$

Description Logics (DLs) [11] are the family of knowledge representation languages that can be used to formally describe knowledge of an application domain (also called Domain of Interests–DoI). DLs have been well studied for many years before being the underlying formalism of Web Ontology Language (OWL) [6] recommended by W3C.

In DLs, knowledge of the Domain of Interests is modeled by a set of *concepts* and *relationships* among those concepts. A simple concept, e.g. *Room*, *Sensor*, is viewed as a set of *individuals* (also called *instances* or *objects*) that share common properties. More complex concepts, e.g. $Room \sqcap \exists has.Sensor$ read as: Room that has some Sensor, are constructed from simple ones using *roles* *has* and *constructors*, e.g., \sqcap and \exists .

A *Knowledge Base* in a Description Logic often consists of two parts: TBox (terminological part) and ABox (assertion part). TBox contains the definition of concepts in a particular application domain, relationships between concepts, and additional constraints over those concepts. ABox is about facts of that domain and specified through *assertions* that relate individuals to concepts and roles.

Description Logic languages are considered as decidable fragments of First Order Logic (FOL) in such a way that concepts and roles correspond to unary and binary predicate respectively, constructors correspond to logical implications. For examples, in \mathcal{ALC} , the most basic language of interest, one can write complex concept ,e.g. $\neg C, C \sqcap D, \exists R.C, \forall R.C$, using negation, conjunction, and quantifiers. $SOIQ(\mathbf{D})$ is an extension to \mathcal{ALC} ; its name indicates the following syntactic constructors:

- \mathcal{S} : transitivity is added to \mathcal{ALC} . We can express that a role is transitive.
- \mathcal{O} : nominal. It allows us to construct a concept using a set of objects.
- \mathcal{I} : inverse roles; r^- is the inverse role of r .
- \mathcal{Q} : Qualified number restriction. We can specify a cardinality constraint on a role with the role filler.
- \mathcal{D} : data types, e.g. String and Integer.

Table 1 shows the syntax and semantics in $SOIQ(\mathbf{D})$. We choose $SOIQ(\mathbf{D})$ because it is expressive enough for SDT and it has a good balance between expressiveness and computational complexity. Moreover, the Ontology Web Language (OWL2) recommended by the W3C is based on $SRIOIQ(\mathbf{D})$, which includes $SOIQ(\mathbf{D})$. We can use various existing tools for editing, e.g., Protégé [7], and reasoning, e.g., Pellet [8] and Hermi [9].

In what follows, we will show how to use $SOIQ(\mathbf{D})$ to formalize semantics within an SDT.

⁶ <http://www.w3.org/tr/owl2-overview>

⁷ <http://protege.stanford.edu>

⁸ <http://clarkparsia.com/pellet/>

⁹ <http://www.hermit-reasoner.com>

Table 1. Syntax and semantics of $SOIQ(D)$

Syntax		Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Top concept	\top	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Bottom concept	\perp	$\perp^{\mathcal{I}} = \emptyset$
Nominal	$\{d\}$	$\{a\}^{\mathcal{I}} = a^{\mathcal{I}}$; where a is an individual
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Transitivity	$trans(S)$	$\{(a, b), (b, c)\} \subseteq S^{\mathcal{I}} \rightarrow (a, c) \in S^{\mathcal{I}}$
Negation	$\neg C$	$(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \text{there is } e \in \Delta^{\mathcal{I}} : (d, e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$
Value restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}} : (d, e) \in R^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$
Qualified number restriction (at least)	$\geq nR.C$	$(\geq nR.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \text{card}(\{e \mid (d, e) \in R^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \geq n\}$
Qualified number restriction (at most)	$\leq nR.C$	$(\leq nR.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \text{card}(\{e \mid (d, e) \in R^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \leq n\}$
		where $\text{card}(X)$ is the cardinality of a set X

3 Formalizing Semantics within a Semantic Decision Table

A decision table has three constituents: *conditions*, *actions* and *decision rules* [3]. Each condition has a *condition stub* and a *condition entry*. A decision rule, which is represented as a table column, is a combination of a set of conditions and actions.

$$\begin{aligned}
\text{ConditionEntry} &\sqsubseteq \forall \text{has.}(\text{string} \sqcup \text{boolean} \sqcup \text{integer} \sqcup \text{float}) \\
\text{Condition} &\sqsubseteq \leq 1 \text{has.} \text{ConditionStub} \sqcap \exists \text{has.} \text{ConditionStub} \\
\text{Condition} &\sqsubseteq \leq 1 \text{has.} \text{ConditionEntry} \sqcap \exists \text{has.} \text{ConditionEntry} \\
\text{ActionEntry} &\sqsubseteq \forall \text{has.}(\text{string} \sqcup \text{boolean}) \\
\text{Action} &\sqsubseteq \leq 1 \text{has.} \text{ActionStub} \sqcap \exists \text{has.} \text{ActionStub} \\
\text{Action} &\sqsubseteq \leq 1 \text{has.} \text{ActionEntry} \sqcap \exists \text{has.} \text{ActionEntry}
\end{aligned}$$

The above axioms describes the basic structure of a decision table according to [3]. A decision rule cannot be modeled as an axiom in DLs, but can be modeled as a query. Table 2 is an example of decision table for a smart home. It is to decide which messages a screen will show and which ring tones an iPhone will ring, depending on whether the ear of a smart rabbit is moved or not, and whether there is pressure on a crib or not. In this example, “People move Ear” is a condition stub. “Yes” is a condition entry. “Screen shows Message” is an action stub. “Message1” is an action entry. Columns 1-4 are the four decision rules. For example, we formalize column 1 as follows:

```

ActionStub(screenShowsMessage),
hasActionEntry(screenShowsMessage, "message1"),
ActionStub(iphoneRings), hasActionEntry(iphoneRings, "")
←
ConditionStub(peopleMoveEar),
hasConditionEntry(peopleMoveEar, yes),
ConditionStub(pressureOnCrib), hasConditionEntry(pressureOnCrib, yes)

```

Table 2. A decision table example used in a ubiquitous system

Condition	1	2	3	4
People move Ear	Yes	No	Yes	No
Pressure on Crib	Yes	Yes	No	No
Action				
Screen shows Message	Message1			
iPhone rings			RingTone1	

A semantic decision table (SDT, [19]) contains three parts: a *decision table*, a set of *lexons* and *commitments*. A lexon is a binary fact type, which has the format of $\langle \gamma, t_1, r_1, r_2, t_2 \rangle$. t_1 and t_2 are the two terms that represent two concepts in a natural language; r_1 and r_2 are the two roles that the two concepts presented by t_1 and t_2 can possibly play with. γ is the context identifier that points to the document where t_1 and t_2 are originally defined, and where r_1 and r_2 become meaningful. A context identifier can be, e.g., a URI. We formalize $\langle \gamma, t_1, r_1, r_2, t_2 \rangle$ as $t_1 \sqsubseteq \forall r_1.t_2$, $t_2 \sqsubseteq \forall r_2.t_1$, and $t_1 = t_2$. In this paper, we will not literally show any lexons seeing that they are anyhow illustrated in the commitments. For instance, the lexon $\langle \gamma, Screen, shows, isShownBy, Message \rangle$ is illustrated in $Screen \sqsubseteq \geq 1shows.Message$, which contains a mandatory constraint on this lexon.

Commitments contain axioms and assertions on lexons. The axioms illustrated in the earlier discussion in this section are the commitments as well. An important characteristic of SDT concerning the commitments is the feasibility of *commitment translation*. In the early literature of SDT, we called it “verbalization”. It is a way to help non-technical decision modelers to understand SDT commitments by providing them with translated sentences in a user-friendly, pseudo-natural language. For example, the commitment $Screen \sqsubseteq \geq 1shows.Message$ can be translated into the sentence “EACH Screen shows AT LEAST ONE Message”.

In the rest of this paper, we will reuse the notations concerning data types from [12] and we adopt the *Unique Name Assumption*, which means we consider two individuals differently when they have different names.

Table 3¹⁰ contains the formalization of the semantics in the SDT for the decision table illustrated in Table 2. In the meanwhile, we show how those commitments are translated into sentences in a pseudo-natural language. As we can see, this SDT is consistent. In this example, we also want to show that a condition stub or an action stub

¹⁰ All the relevant SDT commitments for each SDT in this paper can be downloaded at

www.starlab.vub.ac.be/website/SDT_SOIQ

can be modelled as an instance of a concept (see commitments 1 and 2) or an axiom containing several concepts (see commitments 10-12). What form we should use is dependent on what we need to reason at the end. In the following subsections, we will illustrate inconsistent SDTs and show how to detect the inconsistency.

Table 3. The SDT commitments and their translation for Table 2

ID	DL formalization	Commitment Translation
1	$ConditionStub \sqsubseteq \{peopleMoveEar, pressureOnCrib\}$	$ConditionStub$ is $\{peopleMoveEar, pressureOnCrib\}$
2	$ActionStub \sqsubseteq \{screenShowsMessage, iPhoneRings\}$	$ActionStub$ is $\{screenShowsMessage, iPhoneRings\}$
3	$ActionEntry \sqsubseteq \{message1, ringTone1\}$	$ActionEntry$ is $\{message1, ringTone1\}$
4	$\{peopleMoveEar\} \sqsubseteq \forall hasConditionEntry. boolean$	The VALUETYPE of the $ConditionEntry$ of $peopleMoveEar$ is $boolean$
5	$\{pressureOnCrib\} \sqsubseteq \forall hasConditionEntry. boolean$	The VALUETYPE of the $ConditionEntry$ of $pressureOnCrib$ is $boolean$
6	$Bunny \sqsubseteq \leq 1 has.Name \sqcap \exists has.Name$	EACH $Bunny$ has EXACTLY ONE $Name$
7	$Crib \sqsubseteq \leq 1 has.Name \sqcap \exists has.Name$	EACH $Crib$ has EXACTLY ONE $Name$
8	$Screen \sqsubseteq \geq 1 has.Message$	EACH $Screen$ has AT LEAST ONE $Message$
9	$IPhone \sqsubseteq \geq 1 ringWith.RingTone$	EACH $IPhone$ has AT LEAST ONE $RingTone$
10	$People \sqcap \exists move.Ear \sqsubseteq ConditionStub$	$People$ $move$ Ear is $ConditionStub$
11	$\exists hasPressure.Crib \sqsubseteq ConditionStub$	$Pressure$ on $Crib$ is $ConditionStub$
12	$Screen \sqcap \exists shows.Message \sqsubseteq ConditionStub$	$Screen$ $shows$ $Message$ is $ConditionStub$
13	$IPhone \sqcap \exists ring. \top \sqsubseteq ConditionStub$	$IPhone$ $rings$ is $ConditionStub$
14	$\{screenShowsMessage\} \sqsubseteq \forall hasActionEntry.string$	The VALUE TYPE of the $ActionEntry$ of $screenShowsMessage$ is $string$
15	$\{iPhoneRing\} \sqsubseteq \forall hasActionEntry.string$	The VALUE TYPE of the $ActionEntry$ of $iphoneRing$ is $string$

3.1 Value Constraint

A value constraint, sometimes called *domain constraint*, indicates which values are allowed in a concept (in the case that this concept represents a value type) or role ([5], p. 216-221).

There are three kinds of value constraints: *enumeration*, *range* and *multiple*. With enumeration, we list all the possible values. For example, the commitment “VALUE of Gender is {“M”,“F”}” can be formalized as follows.

$$\top \sqsubseteq \forall hasGender. \{“M”,“F”\}$$

If we can list the values in a continuous order, then we can specify it with a range. The range can as well be unbounded. For example, we have a commitment “VALUE of Age is [0,..)”, which we formalize as $\top \sqsubseteq \forall hasAge.int[\geq 0]$. Multiple value constraint combines enumeration and range.

Table 4 shows an SDT, the commitments of which contain formalization at two different levels - conceptual (e.g., commitments 17-19) and data (e.g., commitments 20-22). More specifically, commitments 18-19 are three axioms, with which we specify value constraints of condition entries, and commitments 20-22 are three assertions, with which we translate conditions from decision columns. For example, the condition $\langle LoginStatus, Maybe \rangle$ in column 3 is translated into the assertion in commitment 20. The condition $\langle TemperatureSensor, (\geq -10, < 0) \rangle$ in column 4 is translated into the assertion shown in commitment 21, to which we randomly assign a value in the value range indicated in the condition entry. And the condition $\langle Age, (\geq 100, \leq 350) \rangle$ in decision column n is translated into the assertion in commitment 22. As we can see, commitment 20 is inconsistent with commitment 19. Commitment 21 is inconsistent with commitment 18. Commitment 22 is inconsistent with commitment 17. Therefore, decision columns 3, 4 and n from Table 4 are inconsistent.

3.2 Cardinality and Occurrence Frequency

A cardinality constraint can be either an *object cardinality* or a *role cardinality* ([5], p. 289). Object cardinality is applied to a lexon term when we want to restrict the number of members or instances of the population of the type that this lexon term points to. For example, if we want to allow at most three X-Box humidity sensors, then we design a commitment as “AT MOST 2 X-Box Humidity Sensors ARE ALLOWED IN ANY CASES” ($\top \sqsubseteq \geq 2 has.XBoxHumiditySensor$).

A role cardinality is comparable to a constraint of occurrence frequency, which is applied when we want to restrict the number of members of the instance of a role. For instance, the commitment “EACH Room has AT MOST 2 X-Box Humidity Sensors” can be formalized as $Room \sqsubseteq \geq 2 has.XBoxHumiditySensor$.

Note that these two commitments are different. The former emphasizes that in any cases, at most two X-Box humidity sensors are allowed; while the latter specifies that only in the case of in a room, at most two X-Box humidity sensors are allowed.

Table 5 shows an example of inconsistency caused by the violation of role cardinality. Column 1 in Table 5 can be translated into the assertions in commitments 26-31. They are inconsistent with commitments 22-25, especially commitment 24.

Note that $Room \sqsubseteq \leq 2 has.XBoxHumiditySensor$ is not enough to fully specify the semantics in this SDT; we need to translate the condition entry “Yes” into, which we link the existence of X-Box Humidity Sensor.

In the following two subsections, we will discuss two specific cases of role cardinality constraints. They are mandatory and uniqueness.

Table 4. An SDT on deciding whether to accept to process or not based on the value received from a temperature sensor, the age and the login state of a user

Condition	1	2	3	4	...	n
Age	≥ 18	≥ 18	≥ 18	≥ 18	...	$\geq 100, \leq 350$
Temperature Sensor	$\geq 0, \leq 30$	$\geq 0, \leq 30$	$\geq 0, \leq 30$	$\geq -10, \leq 0$...	$\geq 0, \leq 30$
Login State	Yes	No	Maybe	Yes	...	Yes
Action						
Accept	*		*	*		*

SDT Commitments

ID	DL Formalization	Commitment Translation
16	$Age \sqsubseteq \forall hasConditionEntry.int[\geq 0 \wedge \leq 200]$	The VALUE RANGE of the <i>ConditionEntry</i> of <i>Age</i> is $[0,200]$ and the VALUE TYPE of the <i>ConditionEntry</i> of <i>Age</i> is <i>Integer</i>
17	$TemperatureSensor \sqsubseteq \forall hasConditionEntry.int[\geq -100 \wedge \leq -20] \sqcup int[\geq 0 \wedge \leq 100]$	The VALUE RANGE of the <i>ConditionEntry</i> of <i>TemperatureSensor</i> is $\{-100, -20\}, [0, 100]$ and the VALUE TYPE of the <i>ConditionEntry</i> of <i>Age</i> is <i>Integer</i>
18	$LoginStatus \sqsubseteq \forall hasConditionEntry.boolean$	The VALUE TYPE of the <i>ConditionEntry</i> of <i>LoginStatus</i> is <i>Boolean</i>
19	$hasConditionEntry(loginStatus, "Maybe")$	The <i>ConditionEntry</i> of <i>loginStatus</i> is "Maybe"
20	$hasConditionEntry(temperatureSensor, -5)$	The <i>ConditionEntry</i> of <i>temperatureSensor</i> is -5
21	$hasConditionEntry(age, 120)$	The <i>ConditionEntry</i> of <i>age</i> is 120

3.3 Mandatory

A lexon role can be mandatory or optional. A mandatory is mandatory iff it is played by every member of the population of its connected object type, otherwise, it is optional ([5], p. 162). A mandatory constraint is equivalent to an "AT LEAST ONE" role cardinality constraint.

Suppose we have a commitment that contains a mandatory constraint, which is "EACH Room has AT LEAST ONE X-Box Humidity Sensor" and formalized as

$$\begin{aligned}
 &Room \sqsubseteq \leq 1 has.XBoxHumiditySensor \\
 &or \\
 &Room \sqsubseteq \exists has.XBoxHumiditySensor
 \end{aligned}$$

Table 5. An SDT on deciding whether or not to turn on Actuator x based on the availability of X-Box557, X-Box120 and MS Xbox 360

Condition	1	2	3	4	5	6	7	8
X-Box 557	Yes	Yes	Yes	Yes	No	No	No	No
X-Box 120	Yes	Yes	No	No	Yes	Yes	No	No
MS XBox 360	Yes	No	Yes	No	Yes	No	Yes	No
Action								
Actuator	*		*		*	*		*

SDT Commitments

ID	DL Formalization	Commitment Translation
22	$XBoxHumiditySensor \sqsubseteq \forall hasConditionEntry. boolean$	The VALUE TYPE of the <i>ConditionEntry</i> of <i>XBoxHumiditySensor</i> is <i>Boolean</i>
23	$XBoxHumiditySensor \equiv \{xBox557, xBox120, msXBox360\}$	<i>XBoxHumiditySensor</i> is $\{xBox557, xBox120, msXBox360\}$
24	$Room \sqsubseteq \leq 2 has.XBoxHumiditySensor \sqcap \exists hasConditionEntry.\{true\}$	EACH <i>Room</i> has AT MOST 2 <i>XBoxHumiditySensor</i> AND the <i>ConditionEntry</i> of <i>XBoxHumiditySensor</i> is <i>true</i>
25	$Room \equiv \{room1\}$	<i>Room</i> is $\{room1\}$
26	$hasConditionEntry(xBox557, true)$	The <i>ConditionEntry</i> of <i>xBox557</i> is <i>true</i>
27	$hasConditionEntry(xBox120, true)$	The <i>ConditionEntry</i> of <i>xBox120</i> is <i>true</i>
28	$hasConditionEntry(mSXbox360, true)$	The <i>ConditionEntry</i> of <i>mSXBox360</i> is <i>true</i>
29	$has(room1, xBox557)$	<i>room1</i> has <i>xBox557</i>
30	$has(room1, xBox120)$	<i>room1</i> has <i>xBox120</i>
31	$has(room1, mSBBox360)$	<i>room1</i> has <i>mSBBox360</i>

Before we apply this commitment to Table 5, we need to add the mapping between “No” and “false” as we did in the previous section. And, we need to translate every decision rule column into a set of assertions. Suppose we want to check column 8 in Table 5. We first modify commitment 24 into the following axiom.

$$Room \sqsubseteq \exists has.(XBoxHumiditySensor \sqcap \neg \forall hasConditionEntry.\{false\})$$

Then, we replace commitment 26-28 with the following assertions.

$$\begin{aligned} &hasConditionEntry(xBox557, false) \\ &hasConditionEntry(xBox120, false) \\ &hasConditionEntry(mSXBox360, false) \end{aligned}$$

As we can see, column 8 is inconsistent.

The above discussed example illustrates how a mandatory constraint can be used when the condition stubs represent value members of an object type and their entries are Boolean values.

Table 6 is another SDT example, which uses a mandatory constraint when an object type is a condition stub and its value members are used as its condition entries.

Table 6. An SDT on deciding whether or not to turn on Actuator x based on the availability of X-Box Humidity Sensors (diverted from Table 5)

Condition	1	2	3
X-Box Humidity Sensor	{X-Box557, X-Box120}	{X-Box557, MS XBox360}	N/A
Actuator	*		*

SDT Commitments		
ID	DL Formalization	Commitment Translation
32	$XBoxHumiditySensor \equiv \{xBox557, xBox120, mSXbox360\}$	$XBoxHumiditySensor$ is $\{xBox557, xBox120, mSXbox360\}$
33	$\{na\} \sqsubseteq \neg XBoxHumiditySensor$	n/a is NOT a XBoxHumiditySensor
34	$XBoxHumiditySensor(na)$	n/a is $XBoxHumiditySensor$

In SDT, if there is no instance of $XBoxHumiditySensor$ (noted as N/A) then the mandatory constraint “EACH Room has AT LEAST ONE X-Box Humidity Sensor” is violated. We map it to the $SOIQ(D)$ axiom as shown in commitment 33, which means that na cannot be an instance of $XBoxHumiditySensor$. Commitment 34 contains an assertion, which is translated from column 3 in Table 6 and it contradicts the axiom in commitment 33.

3.4 Uniqueness

A uniqueness constraint is used when we need to ensure a (co-)role from one lexon or a combination of (co-)roles from several lexons is played at most once. For example, we want to have a uniqueness constraint as “EACH Room has AT MOST ONE X-Box Humidity Sensor”. It is formalized as $Room \sqsubseteq \leq 1 has.XBoxHumiditySensor$. Suppose we apply this constraint on Table 5 and take column 2 in Table 5 as an example. We will get the commitments as shown in Table 7. As illustrated, column 2 is inconsistent.

Similarly, if we translate columns 1, 3 and 5 in Table 5, then we will see that they are also inconsistent. This example shows how we can use a uniqueness constraint with Boolean condition entries.

Table 8 is another example, which shows how to verify decision columns when condition entries are sets. In its commitments, column 3 from Table 8 is formalized. Column 3 is invalid because X-Box120 is a humidity sensor and only one humidity sensor is allowed in one room.

Table 7. SDT commitments containing a uniqueness constraint for Table 5

SDT Commitments		
ID	DL Formalization	Commitment Translation
35	$Room \sqsubseteq$ $\leq 1 has.XBoxHumiditySensor$ $\sqcap \exists hasConditionEntry.\{true\}$	EACH <i>Room</i> has AT MOST ONE <i>XBoxHumiditySensor</i> AND the <i>ConditionEntry</i> of <i>XBoxHumiditySensor</i> is <i>true</i>
36	$hasConditionEntry(xBox557, true)$	The <i>ConditionEntry</i> of <i>xBox557</i> is <i>true</i>
37	$hasConditionEntry(xBox120, true)$	The <i>ConditionEntry</i> of <i>xBox120</i> is <i>true</i>
38	$hasConditionEntry(mSBox360, false)$	The <i>ConditionEntry</i> of <i>mSBox360</i> is <i>false</i>
39	$has(room1, xBox557)$	<i>room1</i> has <i>xBox557</i>
40	$has(room1, xBox120)$	<i>room1</i> has <i>xBox120</i>
41	$has(room1, mSBox360)$	<i>room1</i> has <i>mSBox360</i>

3.5 Exclusive-Or

In an information system, an exclusive-or constraint is used to ensure that two sets do not overlap each other. We can use it to check the combination of *conditions* or *actions*.

Table 9 shows an example containing both situations. Unlike the previous examples, which we translate the condition entry “Yes” into the Boolean value “true”, we use an anonymous individual to specify its existence. For example, we use the commitment *HumiditySensor*(*hs1*) for the condition $\langle HumiditySensor, Yes \rangle$. Column 1 in Table 9 is invalid because the assertions in commitments 53-57 are inconsistent with the axioms in commitments 50-52. Similarly, we can add an exclusive-or relation between Actuator X and Actuator Y. Then column 3 in Table 9 is invalid.

3.6 Subtyping

The “is-a” subtype/taxonomy relationship is probably one of the mostly used ontological relations. A subtype is an object type, each of whose instances belongs to an encompassing type.

We use subtyping to check the validity of a combination of conditions, e.g., Table 10. As *HumiditySensor* is a subtype of *Sensor*, the condition $\langle HumiditySensor, Yes \rangle$ implies that there is a sensor in the room. Therefore, it is impossible to execute a decision rule, which contains the condition $\langle Sensor, No \rangle$. Accordingly, column 2 is invalid.

The formalization of the semantics is illustrated in the SDT commitments in Table 10. Note that we use subset to formalize subtyping. In commitment 58, we specify that *HumiditySensor* is a subtype of *Sensor*. In column 2 from Table 10, we formalize the semantics in the condition $\langle HumiditySensor, Yes \rangle$ into commitment 59,

Table 8. An SDT on deciding whether or not to turn on Actuator x and Actuator y based on the availability of Sensors

Condition	1	2	3
Humidity Sensor	{X-Box557}	{MS Xbox360}	X-Box557
Sensor	{EZEYE 1011A}	{EZEYE 1011A}	X-Box120
Action			
Actuator X	*		*
Actuator Y		*	*

SDT Commitments		
ID	DL Formalization	Commitment Translation
42	$HumiditySensor \equiv \{xBox557, xBox120, msXBox360\}$	$HumiditySensor$ is $\{xBox557, xBox120, msXBox360\}$
43	$Room \sqsubseteq \leq 1 has.HumiditySensor$	EACH $Room$ has AT MOST ONE $HumiditySensor$
44	$Sensor \sqsubseteq \{eZEYE1011A\} \sqcup HumiditySensor$	$Sensor$ is UNION of $\{eZEYE1011A\}$ and $HumiditySensor$
45	$Room(room1)$	$room1$ is $Room$
46	$HumiditySensor(xBox557)$	$xBox557$ is $HumiditySensor$
47	$Sensor(xBox120)$	$xBox120$ is $Sensor$
48	$has(room1, xBox557)$	$room1$ has $xBox557$
49	$has(room1, xBox120)$	$room1$ has $xBox120$

and the one in $\langle Sensor, No \rangle$ into commitment 60. In order to trigger the reasoner to check the inconsistency, we need to have commitment 61, which assigns an anonymous individual to the concept “Room”.

In this section, we have discussed how to formalize the semantics within an SDT, which is used for validating a decision table. In the next section, we will illustrate our related work.

4 Related Work and Discussion

Including the related work concerns the validation issues for decision tables, which we have presented in Sec. 1, it is also important to study the related work concerning the technologies around Semantic Decision Table (SDT).

As discussed, the semantics in an SDT is modeled as ontological commitments, which can be graphically modeled in Semantic Decision Rule Language (SDRule-L, [19]), and which is based on Object Role Modeling language (ORM, [5]). Halpin studied formalization of ORM in first-order logic (FOL) in [4]. Later, Jarrar showed an initial idea on how to map ORM into the \mathcal{DLR}_{idf} (one Description Logic language) in [10] and the \mathcal{SHOIN} Description Logic in [9].

Table 9. An SDT on deciding whether or not to turn on Actuator x and Actuator y based on the availability of Humidity Sensor and Light Sensor

Condition	1	2	3	4
Humidity Sensor	Yes	Yes	No	No
Light Sensor	Yes	No	Yes	No
Action				
Actuator X	*		*	
Actuator Y			*	

SDT Commitments		
ID	DL Formalization	Commitment Translation
50	$Room \sqcap \exists has.HumiditySensor$ $\sqsubseteq RoomWithHS$	Every <i>Room</i> that has a <i>HumiditySensor</i> is called <i>RoomWithHS</i>
51	$Room \sqcap \exists has.LightSensor$ $\sqsubseteq RoomWithLS$	Every <i>Room</i> that has a <i>LightSensor</i> is called <i>RoomWithLS</i>
52	$RoomWithHS \sqcap RoomWithLS$ $\sqsubseteq \perp$	A <i>Room</i> cannot has both <i>HumiditySensor</i> and <i>LightSensor</i>
53	$Room(room1)$	<i>room1</i> is a <i>Room</i>
54	$HumiditySensor(xBox557)$	<i>xBox557</i> is a <i>HumiditySensor</i>
55	$LightSensor(ezEYE1011A)$	<i>ezEYE1011A</i> is a <i>LightSensor</i>
56	$has(room1, xBox557)$	<i>room1</i> has <i>xBox557</i>
57	$has(room1, ezEYE1011A)$	<i>room1</i> has <i>ezEYE1011A</i>

Table 10. An SDT on deciding whether or not to turn on Actuator x based on the availability of Humidity Sensor and Sensor

Condition	1	2	3	4
Humidity Sensor	Yes	Yes	No	No
Sensor	Yes	No	Yes	No
Action				
Actuator x	*	*	*	

SDT Commitments		
ID	DL Formalization	Commitment Translation
58	$HumiditySensor \sqsubseteq Sensor$	<i>HumiditySensor</i> is a subtype of <i>Sensor</i>
59	$Room \sqsubseteq \exists has.HumiditySensor$	<i>Room</i> has SOME <i>HumiditySensor</i>
60	$Room \sqsubseteq \forall has. \neg Sensor$	<i>Room</i> has NO <i>Sensor</i>
61	$Room(room1)$	<i>room1</i> is <i>Room</i>

We have studied how to use FOL to formalize the extension graphical notations, which are not defined in ORM but in SDRule-L, such as sequence and cross-context operators, in [19]. In this paper, we do not focus on those extensions. Instead, we would rather model the ontological commitments using existing ORM constraints. Based on our previous work in [17,20,19] and the related work in [9], we formalize the ontological commitments in $SOIQ(D)$ in this paper.

We want to show how those commitments can be used to reason and validate an SDT. We specify constraints in axioms and decision rules in assertions. Note that a condition or action stub can be a concept or a lexon, which contains two concepts, or an instance. To decide on which decision table element needs to be mapped into what kind of formalization is a trivial task.

5 Conclusion and Future Work

As an extension to decision tables, SDT provide extra advantages while using it for validation:

- It supports multiple decision modelers (also called “decision group”) to create, validate and verify a decision table.
- It contains semantically rich meta-rules for its self-organization
- Its analysis functions take advantages of modern ontology engineering technologies, such as formality, shareability, interoperability and community enhancement.

It is important to ensure the correctness of SDTs especially when a community of decision modelers is involved. This problem belongs to V&V for decision making systems.

In this paper, we have discussed SDT validation issues concerning ontology-based consistency checking. We identify the constraints of value, uniqueness, mandatory, cardinality, exclusive-or and subtyping as the ones that can be directly applied. We formalize the semantics of SDT using $SOIQ(D)$.

The SDT creation method used in this paper is to create an ontology that stores the meta-rules for V&V. This creation phase can be replaced by importing existing ontologies, which requires an extra effort during the annotation process.

By using $SOIQ(D)$, we are able to fully keep the semantics of SDT with constraints mentioned in the paper. The consistency of the resulting ontology can be efficiently checked by current reasoners, which are highly optimized, although consistency checking for $SOIQ(D)$ is NEXPTIME. In our future work, more expressive language with the same computational complexity such as $SHOIQ(D)$ can be used to formalize SDT. By using $SHOIQ(D)$, we will properly model role hierarchies for SDT and make a good use of these hierarchies.

Each SDT commitment contains a DL statement. As discussed, an important characteristic from SDT is the feasibility of commitment translation, with which a non-technical decision modeler can understand the defined semantics. Currently, we are working on the tool, which takes a commitment in the pseudo-natural language as the input, and generates DL statements and an OWL2 file as the output. This tool will be further integrated with the existing SDT tool set.

Acknowledgments. This work has been supported by the Open Semantic Cloud for Brussels (OSCB) project, founded by the Brussels Capital. It's authors' great pleasure to thank Prof. Jan Vanthienen and Prof. Robert Meersman for their valuable discussions for SDT in general.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*, 2nd edn. Cambridge University Press (2007)
2. Boley, H., Tabet, S., Wagner, G.: Design rationale for ruleml: A markup language for semantic web rules. In: Cruz, I.F., Decker, S., Euzenat, J., McGuinness, D.L. (eds.) *SWWS*, pp. 381–401 (2001)
3. CSA. Z243.1-1970 for decision tables, canadian standards association (1970)
4. Halpin, T.A.: *A Logical Analysis of Information Systems: Static Aspects of the Data-oriented Perspective*. PhD thesis, University of Queensland, Australia (1989)
5. Halpin, T.A., Morgan, T.: *Information modeling and relational databases*, 2nd edn. Morgan Kaufmann (2008)
6. Henry Beitz, E., Marselos, N., et al.: *A modern appraisal of decision tables*, a Codasyl report. ACM, New York (1982)
7. Hewett, R., Leuchner, J.H.: Restructuring decision tables for elucidation of knowledge. *Data Knowl. Eng.* 46(3), 271–290 (2003)
8. Ibramsha, M., Rajaraman, V.: Detection of logical errors in decision table programs. *Commun. ACM* 21(12), 1016–1025 (1978)
9. Jarrar, M.: Mapping ORM into the SHOIN/OWL Description Logic. In: Meersman, R., Tari, Z. (eds.) *OTM-WS 2007, Part I. LNCS*, vol. 4805, pp. 729–741. Springer, Heidelberg (2007)
10. Jarrar, M.: Towards Automated Reasoning on ORM Schemes. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) *ER 2007. LNCS*, vol. 4801, pp. 181–197. Springer, Heidelberg (2007)
11. Lew, A.: Optimal conversion of extended-entry decision tables with general cost criteria. *Commun. ACM* 21(4), 269–279 (1978)
12. Motik, B., Horrocks, I.: OWL Datatypes: Design and Implementation. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) *ISWC 2008. LNCS*, vol. 5318, pp. 307–322. Springer, Heidelberg (2008)
13. Pooch, U.W.: Translation of decision tables. *ACM Comput. Surv.* 6(2), 125–151 (1974)
14. Qian, Y., Liang, J., Li, D., Wang, F., Ma, N.: Approximation reduction in inconsistent incomplete decision tables. *Knowl.-Based Syst.* 23(5), 427–433 (2010)
15. Shwayder, K.: Combining decision rules in a decision table. *Commun. ACM* 18(8), 476–480 (1975)
16. Tang, Y.: Towards Using Semantic Decision Tables for Organizing Data Semantics. In: Meersman, R., Dillon, T., Herrero, P. (eds.) *OTM 2010. LNCS*, vol. 6428, pp. 494–503. Springer, Heidelberg (2010)
17. Tang, Y.: Directly Applied ORM Constraints for Validating and Verifying Semantic Decision Tables. In: Meersman, R., Dillon, T., Herrero, P. (eds.) *OTM-WS 2011. LNCS*, vol. 7046, pp. 350–359. Springer, Heidelberg (2011)
18. Tang, Y., Meersman, R.: Use Semantic Decision Tables to Improve Meaning Evolution Support Systems. In: Sandnes, F.E., Zhang, Y., Rong, C., Yang, L.T., Ma, J. (eds.) *UIC 2008. LNCS*, vol. 5061, pp. 169–186. Springer, Heidelberg (2008)

19. Tang, Y., Meersman, R.: SDRule Markup Language: Towards Modeling and Interchanging Ontological Commitments for Semantic Decision Making. In: Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. IGI Publishing, USA (2009) ISBN: 1-60566-402-2
20. Tang, Y., Meersman, R.: Towards Directly Applied Ontological Constraints in a Semantic Decision Table. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 193–207. Springer, Heidelberg (2011)
21. Tang, Y., Meersman, R., Demey, J.: A self-configuring semantic decision table for parameterizing an ontology-based data matching strategy. In: RCIS, pp. 1–9. IEEE (2011)
22. Vanhienen, J., Mues, C., Aerts, A.: An illustration of verification and validation in the modelling phase of kbs development. *Data Knowl. Eng.* 27(3), 337–352 (1998)

Imposing Restrictions over Temporal Properties in OWL: A Rule-Based Approach

Sotiris Batsakis and Euripides G.M. Petrakis

Department of Electronic and Computer Engineering
Technical University of Crete (TUC)
Chania, Greece
{batsakis,petrakis}@intelligence.tuc.gr

Abstract. Introducing the dimension of time in ontologies turns binary relations into ternary which cannot be handled by OWL. Approaches such as the N-ary relations or the 4D-fluents approach discussed in this work, offer satisfactory solutions to this problem. However, data and property semantics are not preserved in the resulting representations nor can they be handled by ordinary reasoners such as Pellet. We propose a rule-based solution to this problem using SWRL.

1 Introduction

Welty and Fikes [3] showed how quantitative temporal information (i.e., in the form of temporal intervals whose left and right endpoints are well defined) as well as, the evolution of concepts in time, can be represented effectively in OWL using the so-called “4D-fluents approach”. In our previous work [1], we extended this approach with qualitative (in addition to quantitative) temporal expressions, allowing for the representation of temporal intervals with unknown endpoints by means of their relation (e.g., “before”, “after”) to other time intervals, or alternatively, by translating relations between temporal intervals into equivalent relations between time instants [2]. Property semantics such as cardinality restrictions or property constraints are not handled. Typically, property semantics in OWL are defined over binary relations. These relations are turned into ternary with the introduction of the temporal dimension. Existing solutions to this problem (e.g., the 4D-fluents or the N-ary relations approach) suggest introducing new objects into the temporal representation and also rewriting ternary relations as sets of binary ones defined between the old and the new objects. Accordingly, property relations, to become meaningful, need to be applied on the new objects as well, rather than on the objects on which they were meant to be defined originally. Then, property semantics can no longer be handled by ordinary reasoners such as Pellet.

In this work, we propose a mechanism for handling OWL property semantics over temporal representations in conjunction with the 4D-fluents and the N-ary relations approaches. Property semantics are expressed by a set of SWRL rules defined over temporal relations (rather than by OWL axioms as it is typical in

static ontologies, since cardinality restrictions over non simple binary properties lead to undecidability [5]). To the best of our knowledge, this is the only known solution to this problem.

Related work in the field of knowledge representation is discussed in Sec. 2. Restriction checking is discussed in Sec. 3, followed by evaluation in Sec. 4 and conclusions and issues for future work in Sec. 5.

2 Background and Related Work

The OWL-Time temporal ontology [1] describes the concepts of time. Apart from language constructs for the representation of time in ontologies, there is still a need for mechanisms for the representation of the evolution of concepts (e.g., events) in time. Representation of the evolution of temporal concepts is achieved using *N-ary relations* [2] or *4D-fluents* [3].

Following the *N-ary relations* approach, the temporal property is represented by two properties, each one related with the new object. Fig. 1(a) illustrates the relation $WorksFor(Employee, Company, TimeInterval)$ representing the fact that an employee works for a company during a time interval using N-ary relations. This approach requires one additional object for every temporal interval. The *4D-fluents* (perdurantist) approach [3] suggests that concepts in time are represented as 4-dimensional objects with the 4th dimension being the time (*timeslices*). Properties having a time dimension are called fluent properties and connect instances of class *TimeSlice*. Fig. 1(b) illustrates the 4D-fluents representation for the temporal relation $Works-For(Employee, Company, TimeInterval)$ of the same example.

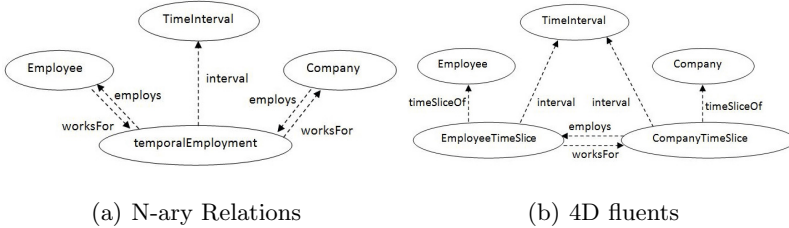


Fig. 1. Example of (a) N-ary Relations and (b) 4D-fluents

2.1 Temporal Ontology

According to Welty and Fikes [3], to add a time dimension to an ontology, classes *TimeSlice* and *TimeInterval* with properties *timeSliceOf* and *timeInterval* are introduced. Properties having a temporal dimension are called *fluent properties* and connect instances of class *TimeSlice* (as in Fig. 1(b)). In [1], the 4D-fluents representation was enhanced with qualitative temporal relations (i.e., relations

¹ <http://www.w3.org/TR/owl-time>
² <http://www.w3.org/TR/swbp-n-aryRelations>

holding between time intervals whose starting and ending points are not specified). A temporal relation can be one of the 13 pairwise disjoint Allen's relations.

Definitions for temporal entities (e.g., instants and intervals) are provided by incorporating OWL-Time into the ontology. A representation based on temporal instants (rather than intervals) is also feasible [2]. Each interval (which is an individual of the *ProperInterval* class of OWL-Time) is related with two instants (individuals of the *Instant* class) that specify its starting and ending points using the *hasBeginning* and *hasEnd* object properties respectively. Notice that, only one of the three relations (*before*, *after*, or *equals*) may hold between any two temporal instants with the obvious interpretation. These relations can be asserted even when exact dates of the time instants are unknown. Relations between intervals are expressed as time instant relations between their starting and ending points.

2.2 Temporal Reasoning

Reasoning is realized by introducing a set of SWRL³ rules operating on temporal relations. Reasoners that support DL-safe rules such as Pellet⁴ can be used for inference and consistency checking over temporal relations. Table 1 represents the result of the composition of two temporal relations pairs (relations *before*, *after* and *equals*, are denoted by symbols “<”, “>”, “=” respectively).

Table 1. Composition Table for point-based temporal relations.

Relations	<	=	>
<	<	<	<, =, >
=	<	=	>
>	<, =, >	>	>

For example, if relation R_1 holds between *instant1* and *instant2* and relation R_2 holds between *instant2* and *instant3*, then the entry of the Table 1 corresponding to line R_1 and column R_2 denotes the possible relation(s) holding between *instant1* and *instant3*. Compositions of relations R_1 , R_2 yielding a unique relation R_3 as a result are expressed in SWRL using rules of the form:

$$R_1(x, y) \wedge R_2(y, z) \rightarrow R_3(x, z)$$

The following is an example of such a temporal composition rule:

$$before(x, y) \wedge equals(y, z) \rightarrow before(x, z)$$

A series of compositions of relations may imply relations which are inconsistent with existing ones (for example the above rule will yield a contradiction if *after*(x, z) has been asserted into the ontology for specific values of x, y, z). Consistency checking is achieved by ensuring path consistency [7] by applying formulas of the form:

³ <http://www.w3.org/Submission/SWRL/>

⁴ <http://clarkparsia.com/pellet/>

$$\forall x, y, k R_s(x, y) \leftarrow R_i(x, y) \cap (R_j(x, k) \circ R_k(k, y))$$

representing intersection of compositions of relations with existing relations (symbol \cap denotes intersection, symbol \circ denotes composition and R_i, R_j, R_k, R_s denote temporal relations). A set of rules defining the result of intersecting relations holding between two instances must also be defined in order to implement path consistency. These rules are of the form:

$$R_1(x, y) \wedge R_2(x, y) \rightarrow R_3(x, y)$$

where R_3 can be the empty relation. For example, the intersection of relations *before* and *after* yields the empty relation, and an inconsistency is detected:

$$\textit{before}(x, y) \wedge \textit{after}(x, y) \rightarrow \perp$$

Path consistency is implemented by defining compositions of relations using SWRL rules and declaring the three basic relations as disjoint. It is sound and complete when applied on the three basic relations [7].

3 Restriction Checking over Temporal Properties

Checking for restrictions holding on time dependent (fluent) properties in the 4D-fluents representation requires particular attention. If a fluent property holds between two objects, then, these objects are only indirectly associated through one or more artificial objects (e.g., *TimeSlice* object in *4D-fluents*). A fluent property is declared between the artificial object and an actual object (as in Figure 1(a)) or between two artificial objects (as in Figure 1(b)). Checking for property restrictions requires adjusting the domain and range of this property from the artificial to the actual objects (e.g., to *Company* and *Employee* objects in Figure 1(a)). For example, for the *worksfor* property in Figure 1(b), the domain of the property is no longer class *Employee* but *timeslice of Employee*. Accordingly, its range is *timeslice of Company*.

Similar adjustments must be made in the case of N-ary relations but in this case, combining transitivity of properties while retaining domain and range restrictions becomes problematic. For example, the *worksfor* relation in Fig. 1(a) must be provided with two alternative domains and ranges. Other restrictions on properties such as symmetry, asymmetry, reflexiveness, irreflexiveness and transitivity can be applied directly on the temporal property retaining the intended semantics.

Universal restrictions (e.g., “all Employees work for a company”) also require adjusting domains and ranges (i.e., all timeslices of employees *workfor* timeslices of companies). Existential restrictions are adjusted as well (if for example each employee must work for some company, then timeslices of employees must work for some timeslices of companies). Notice, that an existential restriction corresponds to an *at least one* qualified cardinality restriction in OWL and the way it is handled is discussed in the rest of this section.

Adjusting cardinality restrictions, functional and inverse functional properties is somewhat more complicated. Functional properties are a special case of

cardinality restrictions (i.e., if a property is functional, then each object must be connected to *at most* one subject which is different for each object). Cardinality restrictions are amenable to two different interpretations depending on the specific application and the intended semantics:

Cardinality restrictions may be interpreted either as restricting the total number of individuals of a class (e.g., *Company*) related with each individual of another class (e.g., *Employee*) through a fluent property at all times or as restricting the number of individuals for each specific temporal interval over which the fluent property holds true. Each interpretation calls for different ways of handling which are discussed in the following.

The first interpretation is handled simply by counting on the number of individuals of a class related to this property and is implemented in SWRL. SWRL rules are applied because OWL cardinality restrictions cannot handle fluent properties connecting objects through intermediate objects. Specifically, imposing cardinality restrictions as OWL axioms in the chain of properties involving the intermediate objects (which are non-simple properties) leads to undecidability [6].

The following rule expresses the restriction that each employee can work for at most n companies. If $n + 1$ company individuals are found to connect with an employee individual, then the restriction is violated (the *Alldifferent* keyword is an abbreviation for a series of axioms imposing that the $n+1$ individuals z_1, z_2, \dots, z_{n+1} are all different). By imposing a *max* cardinality restriction of 0 over property *error* at the definition of class *Employee*, the violation of the cardinality restriction is detected by standard reasoners such as Pellet using the rule:

$$\begin{aligned} & (At - most - rule1)Employee(x) \wedge (tsTimesliceOf(x_1, x) \\ & \wedge \dots \wedge tsTimesliceOf(x_{n+1}, x) \wedge worksfor(x_1, y_1) \wedge worksfor(x_{n+1}, y_{n+1}) \\ & \wedge tsTimesliceOf(y_1, z_1) \dots \wedge tsTimesliceOf(y_{n+1}, z_{n+1}) \\ & \wedge Alldifferent(z_1, z_2, \dots, z_{n+1}) \wedge Company(z_1) \dots \rightarrow error(x, z_1) \end{aligned}$$

An *at-least* restriction is expressed similarly: an *at-most* $n - 1$ rule is applied (changing the asserted property to *satisfies*(x, n)) combined with an *at-least-one* cardinality restriction on the *satisfies* property for class *Employee*. All rules impose also a restriction on the type of objects involved (e.g., they require that only company objects are involved by checking only for objects connected with timeslices of companies). Dropping such a check leads to an unqualified numeric restriction on the property. Notice that the *Open World Assumption* of OWL will cause a reasoner (e.g., Pellet) not to detect an inconsistency of an *at-least* restriction as future assertions might cause invalidity of this inconsistency detection. Instead, the user can retrieve individuals that are not yet proven to satisfy the restriction using the following SPARQL query:

The second interpretation imposes restrictions on the number of individuals associated with an individual of a specific class through a fluent property, for every temporal interval that the property holds true. Checking for such restrictions

```

select distinct ?x
where {
?x rdf:type ex1:Employee.
OPTIONAL{
?x ex1:satisfies ?y.}
FILTER(!bound(?y))}

```

requires applying reasoning rules over the relations between the temporal intervals associated with the fluent property as described in Sec. 2.2. The next step is to detect overlapping and non-overlapping intervals. After the Allen's relations holding between pairs of intervals have been inferred, Allen's properties *during*, *contains*, *starts*, *startedby*, *finishes*, *finishedby*, *overlaps*, *overlapedby*, *equals* are defined as subproperties of property *overlapping*, thus detecting overlapping and non-overlapping intervals. Moreover, properties *before*, *after*, *meets*, *metby* are defined as subproperties of property *non-overlapping*.

Expressing an *at most* restriction for every time interval is based on the following observation: the restriction is violated *if and only if* $n + 1$ distinct individuals are connected with a given individual (through their timeslices) with the relation at hand, and their corresponding intervals are all pairwise overlapping. If $n + 1$ intervals are pairwise overlapping, then, there exists an interval such that $n + 1$ intervals share a common sub-interval, and this can be proven by induction on n . The existence of such an interval implies that for this interval the *at least* restriction is violated. The corresponding rule (used in combination with a cardinality restriction on property error for inconsistency detection by reasoners) is expressed as (the *pairwiseoverlapping* is an abbreviation for a set of *overlapping* relations between all pairs of intervals at hand):

$$\begin{aligned}
& (At - most - rule2)Employee(x) \wedge (tsTimesliceOf(x_1, x)) \\
& \quad \wedge \dots \wedge tsTimesliceOf(x_{n+1}, x) \wedge hasinterval(x_1, w_1) \dots \\
& \quad \wedge hasinterval(x_{n+1}, w_{n+1}) \wedge worksfor(x_1, y_1) \wedge \dots \wedge worksfor(x_{n+1}, y_{n+1}) \\
& \quad \wedge tsTimesliceOf(y_1, z_1) \dots \wedge tsTimesliceOf(y_{n+1}, z_{n+1}) \wedge \\
& \quad Alldifferent(z_1, \dots, z_{n+1}) \wedge pairwiseoverlapping(w_1, \dots w_{n+1}) \\
& \quad \wedge Company(z_1) \dots \rightarrow error(x, z_1)
\end{aligned}$$

As in the case of the first interpretation, the *at-least* restriction cannot be imposed using only SWRL rules due to the Open World Assumption of OWL. Nevertheless, SWRL rules combined with an SPARQL query that detects individuals that are not satisfying the restriction yet, (in a way analogous to the first interpretation) can be applied.

All rules (with both interpretations) involve a time consuming selection of all possible subsets of individuals and intervals. Therefore, expressing restrictions using SWRL may become a tedious task and also detecting inconsistencies can be time consuming. Specifically, rules for imposing a cardinality of *at-least* or *at most* n involves the selection of all combinations of n among k timeslices, or

reified relations, (where k is the number of temporal individuals in the ontology), thus it is not scalable for large values of n . In the case of reification or N-ary relations, cardinality constraints are expressed accordingly, using appropriate adjustments on classes and on properties of objects involved.

Besides representation of cardinality restrictions, value restrictions and adjustments of domains and ranges, the following object property semantics are redefined as follows:

- *Functional*: It is handled as an at most 1 unqualified cardinality restriction.
- *Inverse Functional*: The inverse property is handled as an at most one unqualified cardinality restriction.
- *Symmetric*: The fluent property is *symmetric* too, thus the symmetry axioms apply on the interval that the involved timeslices exist (i.e., the temporal property is declared symmetric).
- *Asymmetric*: This is handled as a cardinality restriction, where the same property cannot hold for interchanged subjects and objects for timeslices that share an overlapping interval.
- *Equivalent*: The fluent properties are equivalent too.
- *Reflexive*: The fluent property is reflexive too; when a timeslice has the property for an interval, it is also the subject of the property for this interval.
- *Irreflexive*: This is handled as a cardinality restriction; two timeslices of an object cannot be related with the property in question if their intervals overlap.
- *Subproperty*: subproperty axioms apply for the fluent properties with the intended semantics.
- *Transitive*: Fluent properties are declared transitive since related timeslices must have equal intervals (by the definition of the 4D-fluent model) and for these intervals transitivity is applied.

Datatype properties have fewer characteristics (i.e., *subproperty*, *equivalence disjointness*, *functional*) and they are handled as is the case of object properties. In the case of N-ary relations, the above adjustments must take into account the different objects involved (i.e., *Events* instead of *timeslices*).

4 Evaluation

The resulting OWL ontology is decidable, since it complies with the corresponding OWL 2 specifications⁵. Introducing the set of temporal qualitative rules of Sec. 2.2 retains decidability since rules are DL-safe rules⁶ and they apply only on named individuals of the ontology Abox using Pellet (which support DL-safe rules). Furthermore, computing the rules has polynomial time complexity since tractable subsets of Allen’s temporal relations are used. Examples of tractable sets include [7]. As shown in [4], by restricting the supported relations set to a

⁵ <http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>

⁶ <http://www.w3.org/TR/rif-rdf-owl>

tractable subset of Allen's interval algebra, path consistency has $O(n^5)$ worst time complexity (with n being the number of intervals) and it is sound and complete. Notice that extending the model to the full set of relations would result into an intractable reasoning procedure.

By applying the *closure method* [4] over Allen's relations, the minimal tractable sets containing the basic relations consist of 29 relations. For this set, the required number of OWL axioms and SWRL rules is 983. An implementation with temporal instants does not require additional relations. Then, the number of OWL axioms and SWRL rules required for reasoning is limited to 20 [2], implying that the representation based on temporal instants must be preferred over the one based on temporal intervals. However, restriction checking mechanism work with either representation. The restriction checking rules are dependent on the specific cardinality restriction imposed (i.e., the complexity of an *at most* r restriction is a parameter of r). Specifically, restriction checking has $O(n^r)$ time complexity.

5 Conclusions and Future Work

We introduce a rule-based approach for handling data and property semantics in temporal ontologies in OWL. Addressing performance and scalability issues for large scale applications are important issues for future research.

Acknowledgement. This research leading to these results has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement No 296170 (Project PortDial).

References

1. Batsakis, S., Petrakis, E.G.M.: SOWL: A Framework for Handling Spatio-Temporal Information in OWL 2.0. In: Bassiliades, N., et al. (eds.) RuleML 2011 - Europe. LNCS, vol. 6826, pp. 242–249. Springer, Heidelberg (2011)
2. Batsakis, S., Stravoskoufos, K., Petrakis, E.G.M.: Temporal Reasoning for Supporting Temporal Queries in OWL 2.0. In: König, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R.J., Jain, L.C. (eds.) KES 2011, Part I. LNCS, vol. 6881, pp. 558–567. Springer, Heidelberg (2011)
3. Welty, C., Fikes, R.: A Reusable Ontology for Fluents in OWL. *Frontiers in Artificial Intelligence and Applications* 150, 226–236 (2006)
4. Renz, J., Nebel, B.: Qualitative Spatial Reasoning using Constraint Calculi. In: *Handbook of Spatial Logics*, pp. 161–215. Springer, Netherlands (2007)
5. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible SROIQ. In: *Proc. KR 2006*, Lake District, UK (2006)
6. Horrocks, I., Sattler, U., Tobies, S.: Practical Reasoning for Expressive Description Logics. In: *Logic for Programming and Automated Reasoning*, vol. 1705, pp. 161–180. Springer, Heidelberg (1999)
7. van Beek, P., Cohen, R.: Exact and approximate reasoning about temporal relations. *Computational Intelligence* 6(3), 132–147 (1990)

OWL RL in Logic Programming: Querying, Reasoning and Inconsistency Explanations*

Jesús M. Almendros-Jiménez

Dpto. de Lenguajes y Computación
Universidad de Almería
04120-Spain
jalmen@ual.es

Abstract. In this paper we describe a logic programming based implementation of the OWL 2 RL fragment. We show how goals are used for querying, forward reasoning permits to infer new knowledge, and ontology inconsistency is handled by backward reasoning, where explanations and (minimal) justifications are given to inconsistent ontologies.

1 Introduction

The OWL 2 specification of the W3C [4] provides OWL profiles that correspond to syntactic subsets of the OWL 2 language, namely OWL EL, OWL QL and OWL RL. OWL RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. OWL RL reasoning systems can be implemented using rule-based engines by forward-chaining. The design of OWL RL was inspired by Description Logic Programs [3] and pD^* [6]. OWL RL is described by a positive Datalog rule set which is a non-trivial superset of RDF(S) rules.

In this paper we describe a logic programming based implementation of the OWL 2 RL fragment. We show how goals are used for querying, forward reasoning permits to infer new knowledge, and ontology inconsistency is handled by backward reasoning, where explanations and (minimal) justifications are given to inconsistent ontologies. The current work is an extension of our previous work [1]. Here, we will focus on how ontology inconsistency is handled by our Prolog based implementation.

OWL RL specification includes rules for inconsistency testing [10]. They have been integrated in our Prolog based implementation of OWL RL in such a way that an error ontology is obtained as an extension of the checked ontology. In other words, the testing an ontology \mathcal{O} generates $\mathcal{O}^I + \mathcal{E}$, where \mathcal{O}^I are the inferred elements from OWL RL rules, and \mathcal{E} are the elements of the error ontology. When the ontology \mathcal{E} (i.e., the instance) is empty, then the ontology is consistent. Using Prolog as query language, the user can inspect the elements of the ontology \mathcal{E} . Moreover, our approach permits to compute the justifications of each

* This work has been supported by the Spanish Ministry MICINN under grant TIN2008-06622-C03-03, Ingenieros Alborada IDI under grant TRA2009-0309, and the JUNTA de ANDALUCÍA (proyecto de excelencia) ref. TIC-6114.

inconsistency. An ontology can be inconsistent due to a bad modeling. Inconsistency comes from unsatisfiable axioms. To eliminate the unsatisfiability one has to nullify all its reasons. A common strategy to do this is to remove axioms from each justification of the unsatisfiability. The resulting ontology is a repair of the unsatisfiability, and the removed axioms is a diagnosis of the unsatisfiability. Therefore, explanations for ontology consequences, and in particular, for inconsistent ontologies, are a key element for ontology repairing. Justifications are minimal explanations, and collect ontology axioms and consequences involved in the reasoning. Root justifications are explanations that are not derived from others. Besides, the user can be interested to inspect derived consequences, called lemmas, to find the reason of an inconsistency. We have included the following services in our Prolog based implementation:

- (a) Computation of each (all) explanation(s) of a given triple.
- (b) Computation of each (all) justification(s) of a given triple.
- (c) Computation of each (all) root justification(s) of a given triple.
- (d) Computation of each (all) lemma(s) of a given triple.

Explanations, justifications and lemmas are obtained by applying backward the OWL RL Prolog rules. Once the ontology has been materialized in secondary memory, certain Prolog goals can be used to obtain the reasons of a given triple. Prolog is used for the traversal of the search tree of the triple and each node of the tree is collected. It permits to collect the explanations. Now, justifications are minimal explanations and root justifications are leaves of the tree, while lemmas are intermediate nodes. The Prolog implementation can be downloaded from our web site <http://indalog.ual.es/OWL-RL-Prolog>. We have tested our implementation with several examples of ontologies including the running example presented. With large ontologies (with millions of triples) we have obtained reasonable benchmarks.

1.1 Related Work

OWL RL has been implemented in some tools for the Semantic Web. The OWLIM tool [2] applies forward chaining to OWL RL rules, having two editions: SwiftOWLIM and BigOWLIM, whereas SwiftOWLIM is an in-memory system, BigOWLIM uses secondary memory. BigOWLIM supports consistency checking with rules without head. The Oracle Database 11g [8] offers a OWL RL implementation based on a materialization of the inferences using forward chaining, using the materialization for query answering. QueryPIE [7] is also an implementation of OWL RL that proposes an hybrid method (forward and backward chaining) in order to improve the performance of query answering. They have studied some optimizations in the application of OWL RL rules such that precomputing of some reasoning branches which appear often in the computation tree and early application of failures branches. Ontology consistency detection, debugging, justification and repairing have been also studied in [9,12,5].

Let us remark that there exists a SWI-Prolog library called *Thea2* [11] to support OWL 2 following precisely its structural syntax specification: every axiom

in the ontology would correspond on a one-to-one basis with facts in the Prolog database. This allows querying the ontology by simply query the Prolog database using goals with variables as arguments. The way in which *Thea2* and our proposal handle OWL 2 is different (OWL representation and rule execution). In addition, our approach has added the handling of inconsistency.

The structure of the paper is as follows. Section 2 will present an example of OWL RL ontology. Section 3 will describe the handling of inconsistency in our approach. Finally, Section 4 will conclude and present future work.

2 OWL RL with Prolog

Let us see an example of an ontology in the OWL RL fragment. Let us suppose an ontology about a social network in which we define ontology classes: *user*, *user_item*, *activity*; and *event*, *message* \sqsubseteq *activity*; and *wall*, *album* \sqsubseteq *user_item*. In addition, we can define (object) properties as follows: *created_by* which is a property whose domain is the class *activity* and the range is *user*, and has two sub-properties: *added_by*, *sent_by* \sqsubseteq *created_by* (used for events and messages, respectively). We have also *belongs_to* which is a (inverse) functional property whose domain is *user_item* and range is *user*; *friend_of* which is a irreflexive and symmetric property whose domain and range is *user*; *invited_to* which is a property whose domain is *user* and range is *event*; *recommended_friend_of* which is a property whose domain and range is *user*, and is the composition of *friend_of* and *friend_of*, but disjoint with *friend_of*; *replies_to* which is an irreflexive property whose domain and range is *message*; *written_in* which is a functional property whose domain is *message* and range is *wall*; *attends_to* which is a property whose domain is *user* and range is *event* and is the inverse of the property *confirmed_by*; *i_like_it* which is a property whose domain is *user* and range is *activity*, which is the inverse of the property *liked_by*. Besides, there are some (data) properties: the content of a message, the date and name of an event, and the nick and password of an user. Finally, we have defined the concepts *popular* which are events *confirmed_by* some user and activities *liked_by* some user: $\text{event} \sqcap \exists \text{confirmed_by.user} \sqsubseteq \text{popular}$ and $\text{activity} \sqcap \exists \text{liked_by.user} \sqsubseteq \text{popular}$ and we have defined constraints: activities are *created_by* at most one user: $\text{activity} \sqsubseteq \leq 1 \text{ created_by.user}$; and *message* and *event* are disjoint classes. Let us now suppose the set of individuals and object/data property instances of Table 1.

From OWL RL reasoning we can deduce new information. For instance, the individual *message1* is an *activity*, because *message* is a subclass of *activity*, and the individual *event1* is also an *activity* because *event* is a subclass of *activity*. The individual *wall_jesus* is an *user_item* because *wall* is a subclass of *user_item*. These inferences are obtained from the subclass relation. In addition, object properties give us more information. For instance, the individuals *message1*, *message2* and *event1* have been *created_by* *jesus*, *luis* and *luis*, respectively, since the properties *sent_by* and *added_by* are sub-properties of *created_by*. In addition, the individual *luis* is a *friend_of* *jesus* because *friend_of* is symmetric. More interesting is that the individual *vicente* is a *recommended_friend_of* *jesus*, because

Table 1. Individuals and object/data properties of the ontology

<i>Ontology Instance</i>
user(jesus), nick(jesus,jalmen), password(jesus,passjesus2011), friend_of(jesus,luis)
user(luis), nick(luis,lamluis), password(luis,luis0000)
user(vicente), nick(vicente,vicente), password(vicente,vicvicvic), friend_of(vicente,luis), i_like_it(vicente,message2), invited_to(vicente,event1), attends_to(vicente,event1)
event(event1), added_by(event1,luis), name(event1,“Next conference”), date(event1,21/10/2012)
event(event2)
message(message1), sent_by(message1,jesus), content(message1,“I have sent the paper”)
message(message2), sent_by(message2,luis), content(message2,“good luck!”), replies_to(message2,message1)
wall(wall_jesus), belongs_to(wall_jesus,jesus)
wall(wall_luis), belongs_to(wall_luis,luis)
wall(wall_vicente), belongs_to(wall_vicente,vicente)

jesus is a friend_of *luis*, and *luis* is a friend_of *vicente*, which is deduced from the definition of recommended_friend_of, which is the composition of friend_of and friend_of. Besides, the individual *event1* is confirmed_by *vicente*, because *vicente* attends_to *event1* and the properties confirmed_by and attends_to are inverses. Finally, there are popular concepts: *event1* and *message2*; the first one has been confirmed_by *vicente* and the second one is liked_by *vicente*.

The previous ontology is consistent. The ontology might introduce elements that make the ontology inconsistent. We might add an user being friend_of of him(er) self. Even more, we can define that certain events and messages are created_by (either added_by or sent_by) more than one user. Also a message can reply to itself. However, there are elements that do not affect ontology consistency. For instance, *event2* has not been created_by users. The ontology only requires to have at most one creator. Also, messages have not been written_in a wall. It cannot be forced by OWL RL.

3 Handling of Inconsistency

The Prolog implementation is based on the representation of OWL by triples, the use of the RDF database of SWI-Prolog, and a bottom-up (i.e., forward chaining) based Prolog interpreter. The Prolog rules for OWL RL have the form *triple():-triple(),...,triple()*. For instance, the rule:

```
triple(C1, rdfs:subClassOf, C3):-triple(C1, rdfs:subClassOf, C2),
triple(C2, rdfs:subClassOf, C3).
```

defines the transitive closure of the subclass relationship. Such a rule loops in a Prolog interpreter. However, with our bottom-up based implementation, the

Table 2. Mapping of OWL RL Inconsistency rules into the Error Ontology

Name	Antecedent	Consequent
<i>eq-diff1</i>	$x \text{ owl:sameAs } y, x \text{ owl:differentFrom } y$	<i>id</i> <i>rdf:type</i> <i>SameAndDifferentIndividuals</i> <i>id</i> <i>individual</i> <i>x</i> <i>id</i> <i>individual</i> <i>y</i>
<i>prp-irp</i>	$p \text{ rdf:type owl:IrreflexiveProperty},$ $x \text{ p } x$	<i>id</i> <i>rdf:type</i> <i>IrreflexiveProperty</i> <i>id</i> <i>property</i> <i>p</i> <i>id</i> <i>individual</i> <i>x</i>
<i>cax-dw</i>	$c_1 \text{ owl:disjointWith } c_2,$ $x \text{ rdf:type } c_1, x \text{ rdf:type } c_2$	<i>id</i> <i>rdf:type</i> <i>DisjointClasses</i> <i>id</i> <i>class</i> c_1 <i>id</i> <i>class</i> c_2 <i>id</i> <i>individual</i> <i>x</i>

process starts from the input ontology (stored by the RDF database) and rules are applied with a forward chaining strategy, where the calls to the predicate **triple** in the rule condition retrieve the instances from the RDF database of the corresponding OWL triple. The inferences are materialized in the RDF database, and we can query the database as follows:

```
?- rdf(X,rdf:type,ex:popular).
X = 'http://www.semanticweb.org...socialnetwork.owl#event1' ;
X = 'http://www.semanticweb.org...socialnetwork.owl#message2'.
false.
?- rdf(X,ex:recommended_friend_of,Y).
X = 'http://www.semanticweb.org/...socialnetwork.owl#jesus',
Y = 'http://www.semanticweb.org/...socialnetwork.owl#vicente' ;
X = 'http://www.semanticweb.org/...socialnetwork.owl#vicente',
Y = 'http://www.semanticweb.org/...socialnetwork.owl#jesus' ;
false.
```

The OWL RL specification includes consistency tests. Such tests are rules in which the head is *false*. Our approach substitutes *false* by a mapping into an ontology called *error ontology*.

In Table 2, we can see examples of the mapping into the error ontology. The error ontology contains two main classes: *Warning* and *Inconsistency*. *Inconsistency* class contains as subclasses: *SameandDifferentIndividuals*, *IrreflexiveProperty*, *AsymmetricProperty*, *DisjointProperties*, *NegativeAssertionProperty*, *EmptyClass*, *DisjointClasses* and *WrongCardinality*, and *Warning* has as subclass *SameIndividuals*. *Warning* and *Inconsistency* elements have an identifier (*id*) and suitable properties. For instance, in rule *eq-diff1* of Table 2, *individual* represents the individuals which are at the same time equal and different; in rule *prp-irp*, *property* is the property found to be reflexive in the value of *individual*; and, in rule *cax-dw*, *class* represents the classes that are not disjoint, and *individual* represents the element that belongs to the intersection.

Now, we would like to show how inconsistency tests are handled by the Prolog implementation. The idea is to extend the set of Prolog rules. For instance, the rule *cax-dw* is represented as follows:

```
triple(Id,rdf:type,'http://www.semanticweb.org/
inconsistency.owl#DisjointClasses'):-
    triple(C1,owl:disjointWith,C2),
    triple(X,rdf:type,C1),
    triple(X,rdf:type,C2),
    gen_id(['error_',X,' belongs to ',C1,' and ',C2],Id).
```

```

triple(Id,inc:class,C1):-triple(C1,owl:disjointWith,C2),
    triple(X,rdf:type,C1),
    triple(X,rdf:type,C2),
    gen_id(['error_',X,' belongs to ',C1,' and ',C2],Id).

triple(Id,inc:individual,X):-triple(C1,owl:disjointWith,C2),
    triple(X,rdf:type,C1),
    triple(X,rdf:type,C2),
    gen_id(['error_',X,' belongs to ',C1,' and ',C2],Id).

....
    
```

where `gen_id` builds a string from a list of strings¹.

Let us suppose that *event1* is a member of the disjoint classes *event* and *message*. In this case, an individual of the class *DisjointClasses* is generated named “*error_event1 belongs to event and message*”, and the elements involved in the inconsistency are represented as follows. The property *class* takes as value the class names *event* and *message*, and *event1* is the value of the property *individual*.

In our Prolog implementation, a message arises whenever an inconsistent ontology is loaded:

```

?- load_owl('socialnetwork.owl').
ONTOLOGY SEEMS TO BE NOT CONSISTENT
true.
    
```

and we can obtain the following information:

```

?- rdf(X,rdf:type,inc:'Inconsistency').
X = 'error_event1 belongs to event and message'.
    
```

Besides, we can inspect the elements related to the inconsistency:

```

?- rdf(X,rdf:type,inc:'Inconsistency'),
    rdf(X,inc:class,C),rdf(X,inc:individual,Z).

X = 'error_event1 belongs to event and message',
C = 'http://www.semanticweb.org/...socialnetwork.owl#event',
Z = 'http://www.semanticweb.org/...socialnetwork.owl#event1' ;
X = 'error_event1 belongs to event and message',
C = 'http://www.semanticweb.org/...socialnetwork.owl#message',
Z = 'http://www.semanticweb.org/...socialnetwork.owl#event1'.
    
```

Our Prolog based implementation is also equipped with the following services:

- (a) Computation of each (all) explanation(s) of a given triple.
- (b) Computation of each (all) justification(s) of a given triple.
- (c) Computation of each (all) root justification(s) of a given triple.
- (d) Computation of each (all) lemma(s) of a given triple.

Let us suppose that we would like to know an explanation of “*event1 has type popular*”. We can make use of the service `explanation` as follows:

¹ We have found very useful for visualization purposes that the individual identifiers of the error ontology are error messages built from variable values and the vocabulary of the rule.

```
?- explanation(ex:event1,rdf:type,ex:popular,S).
S = [rdf(ex:event1, rdf:type, ex:popular)] ;
S = [rdf(ex:event1, rdf:type, ex:somebody_attends), rdf(ex:somebody_attends,
  rdfs:subClassOf, ex:popular)] ;
S = [rdf(ex:somebody_attends, rdfs:subClassOf, ex:popular), rdf(ex:event1,
  rdf:type, exists(ex:confirmed_by, ex:user)), rdf(exists(ex:confirmed_by,
  ex:user), rdfs:subClassOf, ex:somebody_attends)];
...
```

In such a way that Prolog retrieves each one of the explanations of the triple found in the RDF database. However, explanations can be too big in many cases. We have incorporated a service called `justification` to compute minimal explanations of a given triple as follows:

```
?- justification(ex:event1,rdf:type,ex:popular,S).
S = [rdf(ex:event1, rdf:type, ex:popular)] ;
S = [rdf(ex:event1, rdf:type, ex:somebody_attends), rdf(ex:somebody_attends,
  rdfs:subClassOf, ex:popular)] ;
```

For manual repairing the elements of the input ontology can be modified. For this reason, a service, called `root_explanation`, is provided which offers each subset of non derived elements of the ontology that contributes to a given triple. It can be used as follows:

```
?- root_explanation(ex:event1,rdf:type,ex:popular,S).
S = [rdf(ex:somebody_attends, rdfs:subClassOf, ex:popular)] ;
S = [rdf(ex:somebody_attends, rdfs:subClassOf, ex:popular), rdf(ex:vicente,
  rdf:type, ex:user)] ;
S = [rdf(ex:somebody_attends, rdfs:subClassOf, ex:popular), rdf(ex:vicente,
  ex:attends_to, ex:event1), rdf(ex:attends_to, rdfs:domain, ex:user)] ;
....
```

While `root_explanation` provides the non derived ontology elements that contributes to the triple, `root_justification` obtain minimal root explanations:

```
?- root_justification(ex:event1,rdf:type,ex:popular,S).
S = [rdf(ex:somebody_attends, rdfs:subClassOf, ex:popular)] ;
false.
```

In this case there is just one minimal root explanation: the remaining includes this non derived axiom to conclude the triple. However, root justification could not be satisfactory and we can inspect the lemmas (i.e., derived explanations and justifications) associated to a given triple. They can be obtained as follows:

```
?- lemma_justification(ex:event1,rdf:type,ex:popular,S).
S = [rdf(ex:event1, rdf:type, ex:popular)] ;
S = [rdf(ex:event1, rdf:type, exists(ex:confirmed_by, ex:user)), rdf(exists(
  ex:confirmed_by, ex:user), rdfs:subClassOf, ex:popular)] ;
S = [rdf(exists(ex:confirmed_by, ex:user), rdfs:subClassOf, ex:popular), rdf
  (ex:event1, ex:confirmed_by, ex:vicente), rdf(exists(ex:confirmed_by,
  ex:user), owl:onProperty, ex:confirmed_by), rdf(exists(ex:confirmed_by,
  ex:user), owl:someValuesFrom, ex:user)] ;
...
```

4 Conclusions and Future Work

In this paper we have described the elements of a Prolog based implementation for OWL RL, that includes the handling of inconsistent ontologies. As future

work we would like to extend our approach in several directions. First of all, we would like to fully integrate our Prolog-based reasoning tool with the Protégé tool. Currently, Protégé is used for ontology modeling and the OWL RL implementation is triggered at SWI-Prolog command line. We would like to implement a Protégé plugin. Secondly, we would like to extend our work in the line of ontology repairing. In particular, we would like to automatize the repairing of ontologies. Automatic repairing is vital for large ontologies.

References

1. Almendros-Jiménez, J.M.: A Prolog library for OWL RL. In: Proceedings of the 4th International Workshop on Logic in Databases, pp. 49–56. ACM (2011)
2. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: A family of scalable semantic repositories. *Semantic Web* 2(1), 33–42 (2011)
3. Grosz, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: Proc. of the International Conference on World Wide Web, pp. 48–57. ACM Press, NY (2003)
4. McGuinness, D.L., Van Harmelen, F.: OWL 2 Ontology Web Language. Tech. rep. (2009), <http://www.w3.org/TR/owl2-overview/>
5. Horridge, M., Parsia, B., Sattler, U.: Justification Oriented Proofs in OWL. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 354–369. Springer, Heidelberg (2010)
6. ter Horst, H.J.: Extending the RDFS Entailment Lemma. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 77–91. Springer, Heidelberg (2004)
7. Urbani, J., van Harmelen, F., Schlobach, S., Bal, H.: QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 730–745. Springer, Heidelberg (2011)
8. Kolovski, V., Wu, Z., Eadon, G.: Optimizing Enterprise-Scale OWL 2 RL Reasoning in a Relational Database System. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 436–452. Springer, Heidelberg (2010)
9. Moodley, K., Meyer, T., Varzinczak, I.J.: Root Justifications for Ontology Repair. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 275–280. Springer, Heidelberg (2011)
10. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology: Reasoning in OWL 2 RL and RDF Graphs using Rules. Tech. rep. (2009), http://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules
11. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing OWL2 ontologies using Thea: An application of logic programming. In: Proceedings of the 6th International Workshop on OWL: Experiences and Directions, OWLED 2009 (2009)
12. Wu, G., Qi, G., Du, J.: Finding all justifications of OWL entailments using TMS and MapReduce. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, pp. 1425–1434. ACM (2011)

Using Data-to-Knowledge Exchange for Transforming Relational Databases to Knowledge Bases

Tadeusz Pankowski

Institute of Control and Information Engineering,
Poznań University of Technology, Poland
tadeusz.pankowski@put.poznan.pl

Abstract. Mapping relational databases to knowledge bases is a fundamental prerequisite to many data integration activities in databases and in Semantic Web. The crucial issue is then capturing and mapping database integrity constraints in the way that ensures preservation of semantics of data. We propose a set of rules defining dependencies between a relational database and a knowledge base founded on OWL 2 EL. The theory of data exchange is then adapted to develop data-to-knowledge exchange with preservation of data semantics.

1 Introduction

Mapping of relational databases to OWL or RDFS knowledge bases is an important research topic, since it is a necessary precondition for information integration stored in relational databases and in various heterogeneous data repositories on the Web. Also the need to share data among collaborating partners motivates them to expose relational data on the Web in a form of a knowledge base that could be a subject of automatic reasoning procedures. Such a knowledge base should represent the underlying relational database as accurately as possible. Several papers have been published recently on this subject (see the survey [13]). Also the W3C RDB2RDF Working Group is developing a direct mapping standard for translating relational database instances to RDF [2,12]. Primary key- and foreign key database integrity constraints are then identified as crucial issues in defining such mappings.

In this paper, we study the problem of translating relational databases to a knowledge base considering *primary-* and *foreign keys* as well as *not null* integrity constraints. In our approach, we adapt the results of the theory of *data exchange* [6], and *chase procedure* [1]. We identify this problem as the data-to-knowledge exchange (*dk-exchange*) rather than the data-to-data exchange (*dd-exchange*). In general, the dk-exchange problem is much more complex compared to the dd-exchange [3,8]. The reasons are twofold: (1) Databases are based on the *closed world assumption* (CWA) while knowledge bases are based on the *open world assumption* (OWA) [11]. A statement in CWA is treated as false when it is not known to be true. In OWA some new facts can be inferred. In consequence, what

causes an inconsistency in a relational database, can cause an inference of new knowledge in the knowledge base. (2) Databases use unique name assumption while in knowledge bases this assumption is usually not made [4].

The novelties of the paper are the following: 1. We propose a translation of relational database to a set of OWL 2 EL-compatible rules representing the database schema and its instance in a knowledge base. The rules representing the schema and integrity constraints form the set of TBox axioms, and the rules for transforming an instance form a set of data-to-knowledge dependencies (DKDs). 2. We show how the set of TBox axioms should be divided into two sets – standard TBox axioms and integrity constraint TBox axioms. We show that this distinction is necessary in dk-exchange systems to ensure completeness in semantics preservation.

In Section 2 we formulate the problem of dk-exchange. Translation of relational schema along with integrity constraints into a set of knowledge base axioms is proposed in Section 3. In Section 4 the problem of semantics preservation is discussed. Section 6 concludes the paper.

2 Data-to-Knowledge Exchange: Setting of the Problem

2.1 Relational Database

A *relational database schema* (*rdb-schema*) is a pair $\mathcal{R} = (\mathbf{R}, IC)$, where $\mathbf{R} = \{R_1, \dots, R_n\}$ is a *relational schema* and IC is a set of *integrity constraints* over \mathbf{R} . Each *relation symbol* $R \in \mathbf{R}$ has a nonempty finite set $att(R)$ of *attributes*, we assume that $att(R) \cap att(R') = \emptyset$ for $R \neq R'$. An *instance* I of a schema \mathbf{R} is a finite set of facts of the form $R(A_1 : c_1, \dots, A_k : c_k)$, where $R \in \mathbf{R}$, $att(R) = (A_1, \dots, A_k)$, $c_i \in \text{Const} \cup \{\text{NULL}\}$, Const is a set of *constants*. $IC = PKey \cup FKey \cup NotNull$, where: (1) *PKey* – a set of *primary key constraints* of the form $pkey(R, A)$, where $R \in \mathbf{R}$, and $A \in att(R)$. (2) *FKey* – a set of *foreign key constraints* of the form $fkey(R, A, R', A')$, where $R, R' \in \mathbf{R}$, $A \in att(R)$, $A' \in att(R')$; A' must be the primary key of R' , i.e. $pkey(R', A') \in PKey$. (3) *NotNull* – a set of *not null constraints* of the form $notnull(R, A)$, where $R \in \mathbf{R}$, $A \in att(R)$. By $\mathcal{RB} = (\mathbf{R}, IC, I)$ we denote a (relational) *database* with database schema (\mathbf{R}, IC) and an instance I of \mathbf{R} . If I satisfies all integrity constraints in IC , then \mathcal{RB} is a *consistent* otherwise an *inconsistent* database.

Example 1. The following database schema \mathcal{R} defines a university database: $E(EId : v_1, ESId : v_2, Course : v_3, Grade : v_4)$ means that the exam v_1 has been taken by v_2 , the subject of examination was v_3 and the achieved grade was v_4 ; $S(SId : v_1, Faculty : v_2)$ means that a student v_1 is enrolled in the faculty v_2 ; $P(PId : v_1, Name : v_2)$ means that a person v_1 has name v_2 .

$$\begin{aligned} \mathcal{R} &= (\mathbf{R} = \{E(EId, ESId, Course, Grade), S(SId, Faculty), P(PId, Name)\}, \\ &\quad IC = PKey \cup FKey \cup NotNull), \\ PKey &= \{pkey(E, EId), pkey(S, SId), pkey(P, PId)\}, \\ FKey &= \{fkey(E, ESId, S, SId), fkey(S, SId, P, PId)\}, \end{aligned}$$

$$\text{NotNull} = \{\text{nonnull}(\mathbf{E}, \mathbf{EId}), \text{nonnull}(\mathbf{S}, \mathbf{SId}), \text{nonnull}(\mathbf{P}, \mathbf{PId}), \\ \text{nonnull}(\mathbf{E}, \mathbf{ESId}), \text{nonnull}(\mathbf{E}, \mathbf{Course})\}.$$

Foreign keys say that exams were only taken by students, and that each student is a person. The following instance I of \mathbf{R} satisfies all integrity constraints.

$$I_1 : \begin{array}{c} \mathbf{E} \\ \begin{array}{|c|c|c|c|} \hline \mathbf{EId} & \mathbf{ESId} & \mathbf{Course} & \mathbf{Grade} \\ \hline "1" & "3" & "db" & "A" \\ \hline \end{array} \end{array} \quad \begin{array}{c} \mathbf{S} \\ \begin{array}{|c|c|} \hline \mathbf{SId} & \mathbf{Faculty} \\ \hline "1" & "math" \\ \hline "3" & \text{NULL} \\ \hline \end{array} \end{array} \quad \begin{array}{c} \mathbf{P} \\ \begin{array}{|c|c|} \hline \mathbf{PId} & \mathbf{Name} \\ \hline "1" & "john" \\ \hline "2" & "ann" \\ \hline "3" & "eva" \\ \hline \end{array} \end{array}$$

2.2 Knowledge Base

An (extended) *knowledge base* is a tuple $\mathcal{KB} = (\mathbf{N}, \mathcal{S}, \mathcal{C}, \mathcal{A})$ (see [8]), where

1. $\mathbf{N} = \mathbf{N}_C \cup \mathbf{N}_{OP} \cup \mathbf{N}_I$ is the *vocabulary* of \mathcal{KB} , where: \mathbf{N}_C – a set of *class names*, \mathbf{N}_{OP} a set of *object property names*, $\mathbf{N}_I = \text{Tuple} \cup \text{Const} \cup \text{Var}$ – a set of *individual names*, Tuple and Var are infinite sets of *labeled nulls* (denoted \perp_1, \perp_2, \dots) for, respectively, tuples and attribute values. They are used as "fresh" Skolem terms for unknown tuples and unknown attribute values.
2. $\mathcal{S} \cup \mathcal{C}$ is a set of TBox *axioms* divided into two groups [8]: (a) \mathcal{S} is a finite set of *standard* TBox axioms which can infer new facts; (b) \mathcal{C} is a finite set of *integrity constraint* TBox axioms used to check whether a given set \mathcal{A} of facts satisfies conditions specified by \mathcal{C} .
3. \mathcal{A} is a set of ABox *facts* (or *assertions*) of the form: (a) $C(a)$ – a *class assertion*, $C \in \mathbf{N}_C$, $a \in \mathbf{N}_I$, (b) $P(a_1, a_2)$ – an *object property assertion*, $P \in \mathbf{N}_{OP}$, $a_1, a_2 \in \mathbf{N}_I$; (c) $a_1 \approx a_2$ – an *equality assertion* $a_1, a_2 \in \mathbf{N}_I$.

Axioms in $\mathcal{S} \cup \mathcal{C}$ are either: (a) *tuple generating dependencies*, TGDs: $\forall \mathbf{x}, \mathbf{y}. (\varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z}. \psi(\mathbf{x}, \mathbf{z}))$; or (b) *equality generating dependencies*, EGDs: $\forall \mathbf{x}. (\varphi(\mathbf{x}) \Rightarrow x' \approx x'')$, where: $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are tuples of individual variables, $\varphi(\mathbf{x}, \mathbf{y}), \psi(\mathbf{x}, \mathbf{z})$, and $\varphi(\mathbf{x})$ are conjunctions of atomic formulas over \mathbf{N} , and x', x'' are variables in \mathbf{x} .

The semantics of \mathcal{KB} is defined by means of an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \mathcal{I} \rangle$ [4]. \mathcal{I} is a *model* of \mathcal{A} , written $\mathcal{I} \models \mathcal{A}$, if \mathcal{I} satisfies all facts in \mathcal{A} . \mathcal{A} is *consistent* with \mathcal{S} , denoted $\mathcal{A} \models \mathcal{S}$, if every model of \mathcal{A} is also a model of \mathcal{S} . It was argued in [8] that the satisfaction of integrity constraint axioms \mathcal{C} should be treated in another way. \mathcal{C} is satisfied in $\mathcal{A} \cup \mathcal{S}$ if every minimal Herbrand model of Skolemized version of $\mathcal{A} \cup \mathcal{S}$ is also a model of \mathcal{C} , denoted $sk(\mathcal{A} \cup \mathcal{S}) \models_{MM} \mathcal{C}$.

Further on, by $\mathcal{K} = (\mathbf{N}, \mathcal{S}, \mathcal{C})$ we will denote a *kb-schema*. An ABox \mathcal{A} is said to be *consistent* with \mathcal{K} , written $\mathcal{A} \models \mathcal{K}$ if $\mathcal{A} \models \mathcal{S}$, and $sk(\mathcal{A} \cup \mathcal{S}) \models_{MM} \mathcal{C}$. Then $\mathcal{KB} = (\mathbf{N}, \mathcal{S}, \mathcal{C}, \mathcal{A})$ is a *consistent knowledge base*.

2.3 Data-to-Knowledge Exchange

Definition 1. A *data-to-knowledge exchange*, or *dk-exchange*, is a tuple $\mathcal{M} = (\mathbf{R}, \mathbf{IC}, \mathbf{N}, \mathcal{S}, \mathcal{C}, \Sigma)$, where: (a) $(\mathbf{R}, \mathbf{IC})$ is a *rdb-schema*; (b) $(\mathbf{N}, \mathcal{S}, \mathcal{C})$ is a *kb-schema*; (c) Σ is a set of *dk-dependencies*: $\forall \mathbf{v}, \mathbf{u}. (\varphi_d(\mathbf{v}, \mathbf{u}) \Rightarrow \exists x. \psi_k(x, \mathbf{v}))$, where φ_d and ψ_k are conjunctions of atomic formulas over, respectively, \mathbf{R} and \mathbf{N} ; variables in \mathbf{v} and \mathbf{u} range over $\text{Const} \cup \text{Var}$ and x over Tuple .

A dk-exchange is an adaptation of the dd-exchange setting originally proposed in [6]. Dd-exchange is a quintuple $(\mathbf{R}_s, \Sigma_s, \mathbf{R}_t, \Sigma_t, \Sigma)$, and is defined as the problem of transforming data structured under the source schema \mathbf{R}_s into an instance of the target schema \mathbf{R}_t that reflects the source data as accurately as possible and satisfies all target dependencies Σ_t . Thus, it is assumed that all components in dd-exchange are given in advance, and the problem is to investigate the existence of a solution and to find a solution. The solution can be found by means of a chase procedure if the set of target dependencies Σ_t is *weakly acyclic*. Such the solution is called a *canonical universal solution* [6].

In the case of dk-exchange setting we are interested in transforming both a database schema and its instances into a knowledge base. Thus our task consists of two steps:

1. For a given rbd-schema $\mathcal{R} = (\mathbf{R}, IC)$: (a) translate \mathcal{R} into a kb-schema $\mathcal{K} = (\mathbf{N}, \mathcal{S}, \mathcal{C})$; (b) determine a set Σ of dk-dependencies from \mathbf{R} into \mathbf{N} . In result we obtain a dk-exchange system $\mathcal{M} = (\mathbf{R}, IC, \mathbf{N}, \mathcal{S}, \mathcal{C}, \Sigma)$.
2. For an instance I of \mathbf{R} find an ABox \mathcal{A} such that: (a) $(I, \mathcal{A}) \models \Sigma$; (b) if I is consistent with \mathcal{R} , then \mathcal{A} is consistent with \mathcal{K} (soundness of \mathcal{M}); (c) if I is inconsistent with \mathcal{R} , then \mathcal{A} is inconsistent with \mathcal{K} (completeness of \mathcal{M}).

Such an \mathcal{A} , if exists, is called a *solution* for I with respect to \mathcal{M} .

3 Translation of Relational Database Schema into a Relational Knowledge Base Schema

3.1 Creating Vocabulary

Now, we specify rules for translating rbd-schema (\mathbf{R}, IC) into a kb-schema $(\mathbf{N}, \mathcal{S}, \mathcal{C})$, where $\mathbf{N} = \mathbf{N}_C \cup \mathbf{N}_{OP} \cup \mathbf{N}_I$.

1. $\mathbf{N}_I = \text{Tuple} \cup \text{Const} \cup \text{Var}$ (see Section 2.2).
2. **Thing**, **Nothing**, **Tuple**, and **Val** are in \mathbf{N}_C , ($\text{Val} = \text{Const} \cup \text{Var}$ – a set of attribute values).
3. For each $R \in \mathbf{R}$, there is a class name $C_R \in \mathbf{N}_C$ ($C_R \sqsubseteq \text{Tuple}$).
4. For each attribute $A \in \text{att}(R)$, $R \in \mathbf{R}$, there is a class name $C_A \in \mathbf{N}_C$ ($C_A \sqsubseteq \text{Val}$) and an object property name $P_A \in \mathbf{N}_{OP}$ (from C_R to C_A).
5. For each foreign key $fkey(R, A, R', A') \in FKKey$ there is an object property name $F_A \in \mathbf{N}_{OP}$ (from C_R to $C_{R'}$).

3.2 Creating TBox Axioms

By $\mathcal{S} \cup \mathcal{C} = \tau_{\mathcal{S}, \mathcal{C}}(\mathcal{R})$ we will denote a translation of $\mathcal{R} = (\mathbf{R}, IC)$ into a set of TBox axioms. The axioms are listed in Table 1 as DL expressions and FO formulas (universal quantifications in FO formulas are omitted).

Comments: Axioms: (A1) – (A4), (A8), (A9), (A14) are trivial and are satisfied by assumption (construction of Σ in \mathcal{M}) or are consequences of another axioms. The set \mathcal{S} of standard TBox axioms consists of the following six axioms that are either in Table 1 or are derived from some axioms in Table 1:

Table 1. TBox axioms of relational knowledge base ("-" denotes trivial axioms, S – axioms used to derive \mathcal{S} , and C – axioms in \mathcal{C})

	Constraint	DL	FO
A1.	- disjointness	$\text{Tuple} \sqcap \text{Val} \sqsubseteq \perp$	$\text{Tuple}(x) \wedge \text{Val}(x) \Rightarrow \text{FALSE}$
A2.	- $R \in \mathbf{R}$	$C_R \sqsubseteq \text{Tuple}$	$C_R(x) \Rightarrow \text{Tuple}(x)$
A3.	- $A \in \text{att}(R), R \in \mathbf{R}$	$C_A \sqsubseteq \text{Val}$	$C_A(v) \Rightarrow \text{Val}(v)$
A4.	- domain	$\exists P_A. \top \sqsubseteq C_R$	$P_A(x, v) \Rightarrow C_R(x)$
A5.	S range	$\exists P_A. \top \sqsubseteq C_A$	$P_A(x, v) \Rightarrow C_A(v)$
A6.	S $pkey(R, A)$	$C_A \sqsubseteq \leq 1 P_A. C_R$	$P_A(x_1, v) \wedge P_A(x_2, v) \Rightarrow x_1 \approx x_2$
A7.	S $fkey(R, A, R', A')$	$C_A \sqsubseteq C_{A'}$	$C_A(v) \Rightarrow C_{A'}(v)$
A8.	- domain	$\exists F_A. \top \sqsubseteq C_R$	$F_A(x_1, x_2) \Rightarrow C_R(x_1)$
A9.	- range	$\exists F_A. \top \sqsubseteq C_{R'}$	$F_A(x_1, x_2) \Rightarrow C_{R'}(x_2)$
A10.	S chain	$F_A \circ P_{A'} \sqsubseteq P_A$	$F_A(x_1, x_2) \wedge P_{A'}(x_2, v) \Rightarrow P_A(x_1, v)$
A11.	S existence of FK	$\exists P_A. C_A \sqsubseteq \exists F_A. C_{R'}$	$P_A(x_1, v) \Rightarrow \exists x_2. F_A(x_1, x_2)$
A12.	C existence of ref.att.	$\exists P_A. C_R \sqsubseteq \exists P_{A'}. C_{R'}$	$P_A(x_1, v) \Rightarrow \exists x_2. P_{A'}(x_2, v)$
A13.	- $fkey(R, A, R', A')$	$C_R \sqsubseteq C_{R'}$	$C_R(x) \Rightarrow C_{R'}(x)$
A14.	S and $pkey(R, A)$	$P_A \sqsubseteq P_{A'}$	$P_A(x, v) \Rightarrow P_{A'}(x, v)$
A15.	C $nonnull(R, A)$	$C_R \sqsubseteq \exists P_A. C_A$	$C_R(x) \Rightarrow \exists v. P_A(x, v)$

1. $S_1 = (A5)$ – the range of object property P_A is C_A .
2. $S_2 = (A6)$ – object property P_A is a key for class C_R .
3. $S_3 = (A7)$ – C_A is a subclass of $C_{A'}$, if $fkey(R, A, R', A')$ is a foreign key.
4. $S_4 = P_A(x_1, v) \wedge F_A(x_1, x_2) \wedge P_{A'}(x_2, v) \Rightarrow x_2 \approx x_3$ – follows from (A6) and (A10). If $fkey(R, A, R', A')$ is a foreign key, and a tuple x_1 is connected through P_A to v and through F_A to x_2 , then any tuple x_3 connected by $P_{A'}$ to v is equal to x_2 .
5. $S_5 = (A11)$ – if $fkey(R, A, R', A')$ is a foreign key and a tuple x_1 is connected by P_A to some attribute value v , then there exists such a tuple x_2 of class $C_{R'}$ that x_1 is connected to x_2 by the object property F_A .
6. $S_6 = F_A(x_1, x_2) \wedge P_A(x_1, v) \wedge P_{A'}(x_2, v) \Rightarrow x_1 \approx x_2$ – follows from (A6) and (A14). If $fkey(R, A, R', A')$ is a foreign key and $pkey(R, A)$ is a primary key, then tuples x_1 and x_2 connected by F_A are equal if they are connected to the same value v through object properties, respectively, P_A and $P_{A'}$.

The set \mathcal{C} of integrity constraint axioms consists of the following two axioms:

1. $C_1 = (A12)$ – if $fkey(R, A, R', A')$ is a foreign key, and the object property P_A connects a tuple x_1 with a value v , then there exists a tuple x_2 connected with v by the object property $P_{A'}$.
2. $C_2 = (A15)$ – if $nonnull(R, A)$ is a not null constraint, then for each tuple x of class C_R exists such a value v that x is connected with v by the object property P_A .

3.3 Data-to-Knowledge Dependencies

Let $\mathcal{R} = (\mathbf{R}, IC)$ be a database schema. A set Σ of dk-dependencies (DKDs) is a set of implications determining dependencies between instances of \mathbf{R} and a

set of facts in an ABox \mathcal{A} . Every DKD $\sigma \in \Sigma$ has one of the following forms, where: $R \in \mathbf{R}$, A_k is a primary key, A_i is an attribute distinct from A_k :

$$\begin{aligned} R(A_1 : v_1, A_2 : v_2, \dots, A_n : v_n) \wedge v_k = \text{NULL} &\Rightarrow \exists x.C_R(x), \\ R(A_1 : v_1, A_2 : v_2, \dots, A_n : v_n) \wedge v_k \neq \text{NULL} &\Rightarrow \exists x.C_R(x) \wedge P_{A_k}(x, v_k), \\ R(A_1 : v_1, A_2 : v_2, \dots, A_n : v_n) \wedge v_k \neq \text{NULL} \wedge v_i \neq \text{NULL} & \\ \Rightarrow \exists x.C_R(x) \wedge P_{A_k}(x, v_k) \wedge P_{A_i}(x, v_i). & \end{aligned} \quad (1)$$

Each occurrence of $\exists x$ substitutes to x a "fresh" individual from Tuple. Any fact $R(t)$ from a database instance I is mapped to a set of facts in \mathcal{A} . If the primary key of t is NULL, then $R(t)$ is mapped to a singleton $C_R(x) \in \mathcal{A}$. Otherwise, $R(t)$ is mapped to a set $\{C_R(x), P_{A_j}(x, v_j) \mid A_j \in \text{att}(R), v_j \neq \text{NULL}\} \subseteq \mathcal{A}$. Note that no-key attribute value, v_i , is mapped together with the primary key attribute value, v_k , to ensure the proper identification of the relevant tuple x . Values of the primary key A_k are then used to determine equalities between tuples (denoted by x) (see axiom (A6)).

4 Chasing Facts in Knowledge Base

4.1 Dk-Chase Procedure

The proposed method for finding a solution \mathcal{A} for a given instance I , called *dk-chase* procedure, is based on classical *chase procedure* [16]. It was proven in [6] that if the set of dependencies used in chasing is *weakly acyclic* then the solution can be obtained in polynomial time, or it can be decided in polynomial time that the solution does not exist. The solution, if it exists, is then called a canonical universal solution. The dk-chase procedure is a procedure which successively generates facts in \mathcal{A} using dependencies from Σ (assuming that at the start \mathcal{A} is empty) and "repairs" them relative to axioms in \mathcal{S} so that the result satisfies all dependencies and all axioms. The dk-chasing is denoted

$$(I, \emptyset) \xrightarrow{\Sigma \cup \mathcal{S}} (I, \mathcal{A}),$$

meaning that the solution \mathcal{A} will be achieved after a finite set of steps in which dependencies from $\Sigma \cup \mathcal{S}$ are applied. Finally, $(I, \mathcal{A}) \models \Sigma$ and $\mathcal{A} \models \mathcal{S}$.

4.2 Preservation of Semantics

One of the most challenging issues in dk-exchange is to show that the semantics of the source data is not lost by the transformation (the dk-chase procedure). The semantics is defined by the set of database integrity constraints. The preservation of semantics of a dk-exchange system $(\mathbf{R}, IC, \mathbf{N}, \mathcal{S}, \mathcal{C}, \Sigma)$ can be understood in two ways:

1. A dk-exchange system is *sound* w.r.t. *semantics preservation* if every consistent database $\mathcal{RB} = (\mathbf{R}, IC, I)$ is transformed into a consistent knowledge base $\mathcal{KB} = (\mathbf{N}, \mathcal{S}, \mathcal{C}, \mathcal{A})$, i.e.

$$(I \models \mathbf{R} \wedge (I, \mathcal{A}) \models \Sigma \wedge \mathcal{A} \models \mathcal{S} \wedge I \models IC) \Rightarrow sk(\mathcal{A} \cup \mathcal{S}) \models_{MM} \mathcal{C}.$$

2. A dk-exchange system is *complete* w.r.t. *semantics preservation* if every inconsistent database $\mathcal{RB} = (\mathbf{R}, IC, I)$ is transformed into an inconsistent knowledge base $\mathcal{KB} = (\mathbf{N}, \mathcal{S}, \mathcal{C}, \mathcal{A})$, i.e.

$$(I \models \mathbf{R} \wedge (I, \mathcal{A}) \models \Sigma \wedge \mathcal{A} \models \mathcal{S} \wedge I \not\models IC) \Rightarrow sk(\mathcal{A} \cup \mathcal{S}) \not\models_{MM} \mathcal{C}.$$

It can be proven that a dk-exchange system $\mathcal{M} = (\mathbf{R}, IC, \mathbf{N}, \mathcal{S}, \mathcal{C}, \Sigma)$, where $\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ and $\mathcal{C} = \{C_1, C_2\}$ (see Sec. 3.2), is sound and complete w.r.t. semantics preservation. Moreover, if any axiom is moved from \mathcal{C} into \mathcal{S} then \mathcal{M} is sound but not complete. If any axiom is moved from \mathcal{S} into \mathcal{C} then \mathcal{M} is complete but not sound.

Example 2. Let $\mathcal{S}_1 = \mathcal{S} \cup \mathcal{C}$. We will show that $\mathcal{M}_1 = (\mathbf{R}, IC, \mathbf{N}, \mathcal{S}_1, \emptyset, \Sigma)$ is not complete. Let $I_1 = \{E(Eid : "1", ESid : "3", Course : NULL, Grade : NULL), \emptyset, \emptyset\}$ be an (inconsistent) instance of (\mathbf{R}, IC) (Example 1). Then we can consider only two DKD,

$$\begin{aligned} \sigma_1 &= E(Eid : v_1, ESid : v_2, Course : v_3, Grade : v_4) \Rightarrow \exists x.C_E(x) \wedge P_{Eid}(x, v_1), \\ \sigma_2 &= E(Eid : v_1, ESid : v_2, Course : v_3, Grade : v_4) \Rightarrow \exists x.C_E(x) \wedge P_{Eid}(x, v_1) \\ &\quad \wedge P_{ESid}(x, v_2), \end{aligned}$$

from the set Σ of all DKDs (II). Then $(I_1, \emptyset) \xrightarrow{\Sigma \cup \mathcal{S}_1} (I_1, \mathcal{A}_1)$. It can be easily seen that the target knowledge base $\mathcal{KB}_1 = (\mathbf{N}, \mathcal{S}_1, \emptyset, \mathcal{A}_1)$ is consistent, although the source database $\mathcal{RB}_1 = (\mathbf{R}, IC, I_1)$ is not. \mathcal{A}_1 is presented in Figure 1(a) in a form of a graph. Solid lines in the graph denote facts generated by dependencies from Σ and axioms from $\mathcal{S} \subset \mathcal{S}_1$, dotted lines denote facts inferred as consequences of $\mathcal{C} \subset \mathcal{S}_1$. In Figure 1(b) there is depicted the ABox \mathcal{A}_2 as the result of $\mathcal{M}_2 = (\mathbf{R}, IC, \mathbf{N}, \mathcal{S}, \mathcal{C}, \Sigma)$ on I_1 , i.e. $(I_1, \emptyset) \xrightarrow{\Sigma \cup \mathcal{S}} (I_1, \mathcal{A}_2)$. Then $\mathcal{KB}_2 = (\mathbf{N}, \mathcal{S}, \mathcal{C}, \mathcal{A}_2)$ is inconsistent, similarly as \mathcal{RB}_1 . We can show that \mathcal{M}_2 is sound and complete.

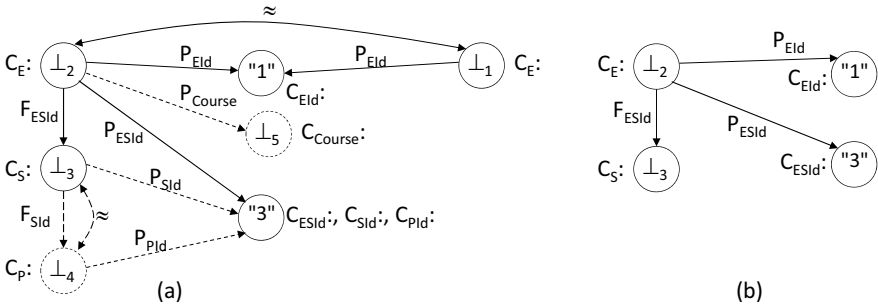


Fig. 1. Consistent ABox \mathcal{A}_1 (a) (w.r.t. $(\mathbf{N}, \mathcal{S}_1, \emptyset)$) produced by \mathcal{M}_1 from inconsistent instance I_1 , and inconsistent ABox \mathcal{A}_2 (b) (w.r.t. $(\mathbf{N}, \mathcal{S}, \mathcal{C})$) produced by \mathcal{M}_2 from I_1

5 Conclusion

In this paper, we study how to map relational databases to OWL 2 EL-based knowledge bases. We consider primary keys, foreign keys, and not-null properties as database integrity constraints and discuss how the relational schema along with the integrity constraints can be represented by vocabulary and TBox axioms of a knowledge base. Next, we apply a chase procedure to define a transformation of a database instance to a set of ABox facts. We focus on the problem of semantics preservation in the data-to-knowledge exchange. We show how the set of TBox axioms should be divided into a set of standard axioms and a set of integrity constraint axioms to achieve complete semantics preservation. This problem is of significant importance in many database-centered and Semantic Web applications. We face it by developing the SixP2P system for semantic integration of data in P2P environment [5,10]. The ontology-based approach is then used to automatic generation of mappings and query rewriting [7,9].

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF (2012), <http://www.w3.org/TR/rdb-direct-mapping>
3. Arenas, M., Botoeva, E., Calvanese, D.: Knowledge Base Exchange. In: 24th International Workshop on Description Logics, DL 2011, pp. 1–11 (2011)
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Petel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
5. Brzykcy, G., Bartoszek, J., Pankowski, T.: Schema Mappings and Agents Actions in P2P Data Integration System. Journal of Universal Computer Science 14(7), 1048–1060 (2008)
6. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
7. Gottlob, G., Orsi, G., Pieris, A.: Ontological Queries: Rewriting and Optimization (Extended Version), pp. 1–25. CoRR, abs/1112.0343 (2011)
8. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. Journal of Web Semantics 7(2), 74–89 (2009)
9. Pankowski, T.: Combining OWL Ontology and Schema Annotations in Metadata Management. In: Corchado, E., Kurzyński, M., Woźniak, M. (eds.) HAIS 2011, Part I. LNCS, vol. 6678, pp. 255–262. Springer, Heidelberg (2011)
10. Pankowski, T.: Pattern-Based Schema Mapping and Query Answering in Peer-to-Peer XML Data Integration System. In: Yan, L., Ma, Z. (eds.) Advanced Database Query Systems: Techniques, Applications and Technologies, ch. 9, pp. 221–246. IGI Global (2011)
11. Reiter, R.: On Closed World Data Bases. In: Logic and Data Bases, pp. 55–76 (1977)
12. Sequeda, J., Arenas, M., Miranker, D.P.: On Directly Mapping Relational Databases to RDF and OWL (Extended Version). CoRR, abs/1202.3667 (2012)
13. Sequeda, J., Tirmizi, S.H., Corcho, Ó., Miranker, D.P.: Survey of directly mapping SQL databases to the Semantic Web. Knowledge Eng. Review 26(4), 445–486 (2011)

PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners

Gen Zou¹, Reuben Peter-Paul¹, Harold Boley^{1,2}, and Alexandre Riazanov³

¹ Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
gen.zou@unb.ca, reuben.peterpaul@gmail.com

² National Research Council of Canada
harold.boleynrc.gc.ca

³ Department of Computer Science and Applied Statistics, UNB, Saint John, Canada
alexandre.riazanov@gmail.com

Abstract. PSOA RuleML is a recently specified rule language combining relational and object-oriented modeling. In order to provide reasoning services for PSOA RuleML, we have implemented a reference translator, PSOA2TPTP, to map knowledge bases and queries in the PSOA RuleML presentation syntax (PSOA/PS) to the popular TPTP format, supported by many first-order logic reasoners. In particular, PSOA RuleML reasoning has become available using the open-source VampirePrime reasoner, enabling query answering and entailment as well as consistency checking. The translator, currently composed of a lexer, a parser, and tree walkers, is generated by the ANTLR v3 parser generator tool from the grammars we developed. We discuss how to rewrite the original PSOA/PS grammar into an *LL(1)* grammar, thus demonstrating that PSOA/PS can be parsed efficiently. We also present a semantics-preserving mapping from PSOA RuleML to TPTP through a normalization and a translation phase. We wrap the translation and querying code into RESTful Web services for convenient remote access and provide a demo Web site.

1 Introduction

Semantic Web knowledge representations span objects, rules, and ontologies. PSOA RuleML [1] is a positional-slotted object-applicative rule language, including light-weight ontologies, which integrates relations (predicates) and objects (frames). To test the PSOA/PS syntax specification and also illustrate the PSOA RuleML semantics, we have developed a *reference implementation* as a translator named PSOA2TPTP [2] mapping knowledge bases and queries of PSOA RuleML in RIF-RuleML-like Presentation Syntax (PSOA/PS) to the TPTP [3] format – a *de facto* standard supported by many first-order logic reasoners. The translated document can then be executed by the open-source first-order reasoner VampirePrime [3] or other TPTP systems for query answering or related

¹ <http://psoa2tptp.googlecode.com/>

² Thousands of Problems for Theorem Provers, <http://www.cs.miami.edu/~tptp/>

³ <http://riazanov.webs.com/software.htm>

reasoning tasks, such as consistency testing and entailment checking (theorem proving).

The main components of our two realizations of PSOA2TPTP include a shared lexer and parser as well as two tree walkers. The lexer breaks the input document up into a stream of tokens. The stream is then transformed by the parser into an Abstract Syntax Tree (AST) which condenses and structures the information of the input. Finally, the AST is traversed by the tree walkers, either for direct TPTP generation or via Abstract Syntax Objects (ASOs). These components are produced by the widely used ANTLR v3 framework⁴ from, respectively, the specified grammars, namely a lexer grammar, a parser grammar and two tree grammars.

To prove feasibility of efficient parsing, we rewrite the original PSOA RuleML EBNF grammar into an $LL(1)$ grammar which accepts a slightly restricted subset of the PSOA RuleML language, including some syntactic sugar proposed in [1]. With this rewriting, the grammar is accepted by ANTLR and is more efficient for parsing. However, it becomes less readable and reusable. Thus, we chose to construct a customized AST by embedding additional rewrite rules into the parser grammar, and to develop an understandable and reusable tree grammar for ANTLR to generate the tree walkers.

To combine and deploy the above-mentioned components, we have also developed a RESTful Web API for translating PSOA/PS documents to TPTP and running VampirePrime. We have published a Web site to demonstrate the use of the API, constituting the first PSOA RuleML implementation release⁵.

The rest of the paper is organized as follows. Section 2 explains the translation source and targets of PSOA2TPTP. Section 3 shows the overall translation architecture. Section 4 discusses grammar implementation, especially the rewriting of the parser grammar. Section 5 gives the syntactic translation from PSOA/PS to TPTP-FOF. Section 6 discusses the RESTful Web API implementation. Section 7 concludes the paper and discusses future work.

2 Interoperation Source and Targets

We discuss here the source language, PSOA RuleML, the target language TPTP, and the target reasoner VampirePrime, of our interoperating translator.

2.1 PSOA RuleML

PSOA RuleML is a rule language that generalizes the POSL [2] as well as the F-Logic and W3C RIF-BLD languages [3]. In PSOA RuleML, the notion of a positional-slotted, object-applicative (psoa) term is introduced as a generalization of: (1) the positional-slotted term in POSL and (2) the frame term and the

⁴ ANother Tool for Language Recognition, a language framework for constructing recognizers, interpreters, compilers and translators from grammatical descriptions containing actions in a variety of target languages. <http://www.antlr.org/>

⁵ <http://198.164.40.211:8082/psoa2tptp-trans/>

class membership term in RIF-BLD. A psoa term has the following general form:

$$o \# f ([t_{1,1} \dots t_{1,n_1}] \dots [t_{m,1} \dots t_{m,n_m}] p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$$

Here, *o* is the object identifier (OID) which gives a unique identity to the object described by the term by connecting three kinds of information: (1) The class membership *o # f* makes *o* an instance of class *f*; (2) each tupled argument $[t_{i,1} \dots t_{i,n_i}]$ represents a sequence of terms associated with *o*; (3) each slotted argument $p_i \rightarrow v_i$ represents a pair of an attribute p_i and its value v_i associated with *o*.

A psoa term can be used as an atomic formula. Atomic formulas in PSOA RuleML can be combined into more complex formulas using constructors from the Horn-like subset of first-order logic: conjunction, disjunction in premises, as well as certain existential and universal quantifiers. Implication can be used to form rules.

2.2 TPTP and VampirePrime

TPTP (Thousands of Problems for Theorem Provers) [10] is a library of test problems for automated theorem proving systems using a problem format of the same name. A TPTP problem is a list of annotated formulas of the form:

$$language(name, role, formula, source, useful\ info).$$

Here, *language* specifies the TPTP dialect used to write the formula. We employ the FOF dialect which allows the use of arbitrary first-order formulas. *name* is a name given to the formula; *role* specifies the intended use of the formula. The most important roles are *axiom*, *hypothesis*, *conjecture* and *theorem*. *formula* is the formula body. *source* and *useful info* are optional and irrelevant for us. Some of the constructors of TPTP are shown in Table 1.

Table 1. TPTP constructors

Symbol	Logical Meaning	Symbol	Logical Meaning
~	not	!=	unequal
&	and	=>	implication
	or	?[v1, v2, ...]	existential quantifier
=	equal	![v1, v2, ...]	universal quantifier

Following is an example of an annotated TPTP formula.

```
fof(first_order, axiom,
    ! [X]: ( (p(X) | ~q(a)) => ?[Y, Z]: (r(f(Y), Z) & ~s) )
).
```

This formula represents the first order formula

$$\forall X : ((p(X) \vee \neg q(a)) \rightarrow \exists Y \exists Z : r(f(Y), Z) \wedge \neg s)$$

Vampire [4] is a mature high-performance reasoner for first-order logic. VampirePrime is an open source reasoner derived from the Sigma KEE⁶ edition of Vampire. In addition to the standard first-order logic theorem proving tasks, such as consistency checking and entailment, VampirePrime supports query answering by implementing incremental query rewriting [5]. It can also be used for semantic querying on relational (SQL) databases modulo arbitrary first-order logic axioms.

3 Translation Architecture

The overall architecture of PSOA2TPTP shown in Figure 1 includes three realizations: the direct translator, the TPTP-Abstract-Syntax-Object-based (TPTP-ASO-based) translator, and the fully-ASO-based translator. We have completed the first two for a subset of the PSOA RuleML language and the last one will be completed in the future. The input for the direct translator is a PSOA/PS document. We use the ANTLR v3 tool to generate a parser-translator that parses the input rulebase and query and generates a semantics-preserving TPTP document, which can be fed into VampirePrime to compute the query results. The concrete steps will be explained later. In contrast to the direct translator, the fully-ASO-based translator will create and transform PSOA RuleML Abstract Syntax Objects (PSOA ASOs) – simple data structures representing the information of the input document in a straightforward manner. The key component is the PSOA-ASO-to-TPTP-ASO translator, which transforms a PSOA

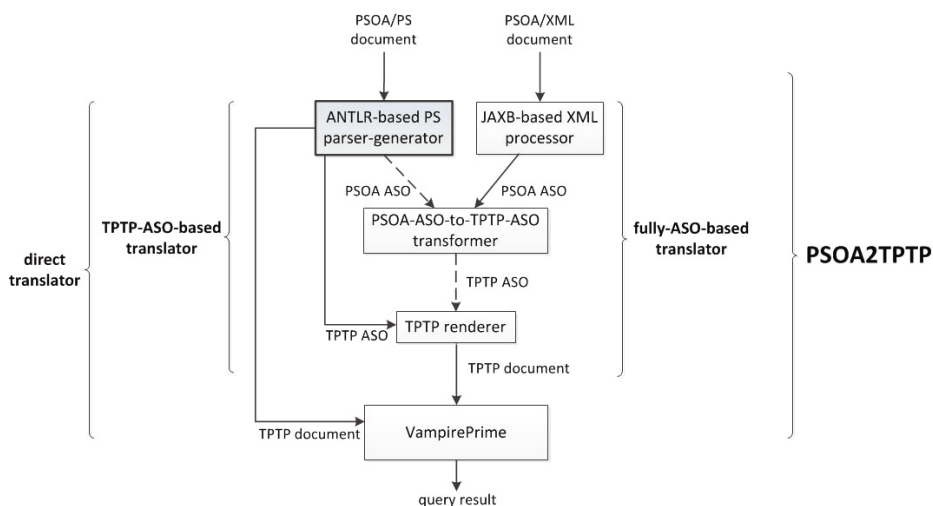


Fig. 1. Components and workflow of the PSOA2TPTP architecture

⁶ http://en.wikipedia.org/wiki/Sigma_knowledge_engineering_environment

ASO into a TPTP ASO using the TPTP parser/renderer library.⁷ The fully-ASO-based translator will also support documents conforming to a PSOA/XML syntax designed in the companion effort PSOA RuleML API [\[6\]](#), which also has developed a JAXB-based⁸ XML parser. The dashed lines in the diagram give the context of components under development. The TPTP-ASO-based translator generates TPTP ASO directly using ANTLR.

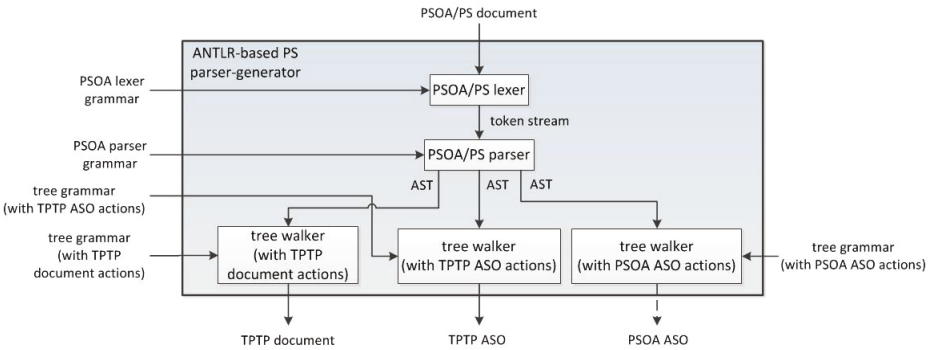


Fig. 2. Workflow of the parsing code generated from the ANTLR grammars

Figure [2](#) ‘explodes’ the shaded box of Figure [1](#), showing the detailed workflow of the ANTLR-based PSOA/PS processor (a parser-generator). Firstly, the input PSOA/PS document is broken up into a token stream by the PSOA/PS lexer. In the stream, every token has an associated regular expression representing all the strings that will be accepted as this token, and the decomposition is done by matching these regular expressions. After this step, the PSOA/PS parser feeds off the token stream, parses its syntactic structure, and creates an Abstract Syntax Tree (AST), which is a condensed version of the input with a tree data structure. Finally, the AST is processed by a tree walker which generates the output. The tree walker on the left-hand side generates the TPTP document directly, which is part of the direct translator in Figure [1](#). The tree walker in the middle (resp., on the right-hand side) creates a TPTP (resp., PSOA) ASO, which is part of the TPTP-ASO-based (resp., fully-ASO-based) translator. The lexer, parser, and three tree walkers are generated by ANTLR from the corresponding ANTLR grammars. The syntactic specifications of the three tree grammars are identical while the embedded output-creating actions in the tree grammars are different. The tree grammar with PSOA ASO actions has not yet been developed while the two other ones have been completed.

Comparing the direct translator and the fully-ASO-based translator, the main advantages of the direct translator are: (d1) It requires fewer steps in Figure [1](#); (d2) it is only based on Java and ANTLR; (d3) it is more efficient, since there are no intermediate representations.

⁷ <http://riazanov.webs.com/tptp-parser.tgz>

⁸ <http://jaxb.java.net/>

The advantages of the fully-ASO-based translator are: (f1) It reuses an existing parser/renderer library for TPTP concrete syntax generation; (f2) the TPTP document generated from the TPTP library is more readable; (f3) it can be reused for the translation from other concrete PSOA syntaxes, such as the PSOA/XML serialization, provided that the corresponding parsers are able to generate PSOA ASOs; (f4) it will be easier to evolve the translation procedure, e.g., by implementing different translation styles, because the developers can work with a clean and simple ASO API instead of the complicated ANTLR AST parsing.

The TPTP-ASO-based translator is an intermediate implementation combining the advantages (d1), (d2), (f1), and (f2). To cover the entire interoperability space, we started with the direct translation, are now developing the TPTP-ASO-based translation, and will then proceed to the PSOA-ASO-based translation, thus allowing an overall comparison.

4 Grammar Implementation for PSOA RuleML Presentation Syntax

In this section, we discuss the implementation of the ANTLR grammars, especially the parser grammar. Firstly, we review the original ANTLR parser grammar for PSOA/PS which, in particular, incorporates the syntactic sugar into the EBNF grammar specification. After that, we propose some additional restrictions on PSOA/PS to make efficient parsing possible. Then we apply some rewriting techniques to the parser grammar to make it acceptable for ANTLR. Finally we discuss the construction of AST and the tree grammar.

4.1 Original Parser Grammar

In [1], the EBNF grammar of PSOA/PS is specified. Besides this core grammar, there is some optional syntactic sugar for psqa terms:

- In the anonymous version of a psqa term, the OID and the hash symbol ‘#’ are omitted.
- For psqa terms without tuples or slots, the empty pair of parentheses can be omitted. For example, the psqa term $o \# f()$, which represents just a membership relation, can be abridged to $o \# f$.
- For psqa terms with only one tuple, the square brackets ‘[’ and ‘]’ enclosing the tuple’s term sequence can be omitted, e.g., $o \# f([t_1 \ t_2] \ p \rightarrow v)$ can be abridged to $o \# f(t_1 \ t_2 \ p \rightarrow v)$.

In order to incorporate this syntactic sugar into the parser grammar, we need to change the original production (1) for psqa term into productions (2) and (3) shown below, where all the productions are in the ANTLR grammar style:

`psoa : term '#' term '(' tuple* (term '->' term)* ')';` (1)

`psoa : term '#' term '(' tuples_and_slots ')'?` (2)
`| term '(' tuples_and_slots ')';`

`tuples_and_slots : tuple* (term '->' term)*` (3)
`| term+ (term '->' term)* ;`

The productions (2) and (3) will be further rewritten in Section [4.3](#).

4.2 Restricted PSOA RuleML Language

In order to simplify rewriting of the original PSOA/PS grammar into one which can be accepted by ANTLR, we impose the following restrictions on the use of the PSOA RuleML language.

1. The ‘-’ character is not allowed in constant and variable names.
2. The class term in a psOA term must be a constant or variable.
3. A subclass formula, an equality formula or an anonymous psOA term must not start with an external term.

The first restriction is introduced to simplify tokenizing of the lexer. The second and third restrictions are brought in when we rewrite the original parser grammar into an *LL*(1) grammar, which will be elaborated in the next section.

4.3 Grammar Rewriting

The ANTLR-generated parser uses an *LL* parsing mechanism. It constructs a DFA (Deterministic Finite Automaton) which can look ahead an arbitrary number of lexer tokens and choose to match one of the candidate patterns in a production. However, the original PSOA RuleML grammar is a non-*LL* grammar and cannot be used directly by ANTLR to generate the parser. So we rewrite it into an *LL*(1) grammar, accepted by ANTLR. The grammar is efficient in that a single-token lookahead tells the parser which alternative to consider. We follow the formal process in the compiler theory to do the rewriting [\[7\]](#): (1) ambiguity resolution; (2) elimination of left recursion; (3) left-factoring.

Ambiguity Resolution. In the original parser grammar, the production for psOA term shown in Section [4.1](#) is ambiguous. The term `o#f()` can be accepted in two ways: (1) `o` is accepted as the OID and `f` as the class term; (2) `o#f` is accepted as a class term and `o#f()` as an anonymous psOA term. To resolve the ambiguity, we restrict the class term to be either a constant or a variable. With this restriction, a higher-order psOA term like `a#b#c` in which `b#c` is the class term needs to be expressed as a conjunction of separate psOA terms `a#b` and `b#c`.

Elimination of Left Recursion. Left recursion is one of the main causes of a grammar to become a non-*LL* grammar. A simple example of a left-recursive grammar with a single terminal *A* is:

$$p : p A \mid ;$$

This grammar of non-terminal *p* accepts a string of the form *A**. However, no *LL* – *based* parser is capable of parsing such a production since it would not be able to consume any token when it applies the alternative $p : p A$.

In PSOA/PS, the production for *psoa* is implicitly left-recursive since its first non-terminal *term* can also be a *psoa*. In order to eliminate left recursion, we employ a rewriting in the following steps, where productions (4), (5) and (8) are the results:

1. Separate *psoa* from the production of *term*.

$$\text{term} : \text{psoa} \mid \text{non_psoa_term} ; \quad (4)$$

$$\text{non_psoa_term} : \text{const} \mid \text{var} \mid \text{external_term} ; \quad (5)$$

2. Merge the two alternatives of the *psoa* production in Section 4.1 by combining the common prefix *term*, and group the remaining part using a new non-terminal *psoa_rest*, yielding production (6). Then we separate the left-recursive part from (6) and get (7).

$$\text{psoa} : \text{term} \text{psoa_rest} ; \quad (6)$$

$$\text{psoa} : \text{non_psoa_term} \text{psoa_rest} \mid \text{psoa} \text{psoa_rest} ; \quad (7)$$

3. Rewrite (7) to remove left recursion.

$$\text{psoa} : \text{non_psoa_term} \text{psoa_rest}^+ ; \quad (8)$$

Left Factoring. After removing the left recursion, the third step of rewriting is left factoring, which makes the grammar an *LL*(1) grammar. Left factoring means to combine multiple alternatives into one by merging their common prefix. Following is an example consisting of non-terminals *p, q* and terminals *A, B*:

$$p : q A \mid q ;$$

$$q : B^+ ;$$

While parsing a sentence of *p*, the parser needs to reach the end of the string to decide on the two alternatives which have a common prefix *q*. Since *q* can match an arbitrary number of tokens, no *LL*(*k*) parser is able to distinguish the alternatives of *p*. By merging the common prefix, we can rewrite the production into $p : q A?$ and the grammar becomes an *LL*(1) grammar.

One of the examples of common prefixes in the original parser grammar is the production for *tuples_and_slots* in Section 4.1 which matches zero or more tuples or slots. The two alternatives have a common prefix *term* which accepts an arbitrary number of lexer tokens. Apart from this, an *LL*-parser is also incapable of predicting the end of *term*⁺ since the start of a slot is also a *term*.

We follow the steps below to rewrite the production into $LL(1)$:

1. Separate the scenario which has the prefix `term` from the first alternative. That is the case where `tuples_and_slots` matches one or more slots.

```
tuples_and_slots : tuple+ (term '->' term)*
                  | term+ (term '->' term)*
                  | (term '->' term)+
                  |
                  ;
```

2. Rewrite the second and third alternatives to separate the prefix `'term'`.

```
tuples_and_slots : tuple+ (term '->' term)*
                  | term+ (term '->' term (term '->' term)*)?
                  | term '->' term (term '->' term)*
                  |
                  ;
```

3. Merge the second and the third alternatives into a single alternative.

```
tuples_and_slots : tuple+ (term '->' term)*
                  | term+ ('->' term (term '->' term)*)?
                  |
                  ;
```

After merging, we can see that the common prefixes between different alternatives are eliminated and the grammar becomes an $LL(1)$ grammar.

Besides the example shown above, there are many other occurrences of common prefixes in the original parser grammar. Some of them relate to multiple productions and rewriting is more difficult. One method we employ to simplify rewriting is adding restrictions to some alternatives to make it easier to separate a common prefix. For example, the third restriction we introduced in Section 4.2 prohibits some use cases of external terms in an atomic formula. It allows us to separate the cases with the prefix `external_term` from `atomic` and combine it with the alternative `formula : external '(' atom ')'`.

4.4 Abstract Syntax Tree and Tree Grammar

In the previous section we have illustrated the rewriting of the parser grammar. The resulting $LL(1)$ grammar is easier for generating the parser but tends to be less reusable and more difficult for future developers to read and work on. Thus, we chose to construct an Abstract Syntax Tree (AST) by embedding rewrite rules into the parser grammar and develop a simple and reusable tree grammar for traversing the AST. Examples of rewrite rules used can be found in [8]. The AST retains the meaningful input tokens and encodes them into a tree structure, while some auxiliary tokens like `'('` and `')'` are removed. Some *imaginary tokens*, in the ANTLR terminology, are also added for easy recognition and navigation.

5 PSOA-to-TPTP Translation

5.1 Semantics-Preserving Translation from PSOA RuleML to TPTP

The semantics-preserving translation from PSOA RuleML to TPTP has two phases: (1) Normalization of composite formulas into a conjunction of elementary constructs and (2) translating them into corresponding TPTP forms.

Normalization. In the normalization phase, we transform the original knowledge base (KB) into a semantically equivalent one which only uses elementary constructs. An elementary construct is a term or a formula that cannot be split into equivalent subformulas. The reason for normalization is that the subsequent one-to-one translation of elementary constructs into TPTP avoids adding additional axioms to derive subformulas. For example, the psoa formula $\text{o\#f}(t_1 \dots t_k)$ is equivalent to a conjunction of two translation-ready subformulas: $\text{o\#f}()$ and $\text{o\#Top}(t_1 \dots t_k)$, where **Top** is the root class such that o\#Top is true for any o . If we translated $\text{o\#f}(t_1 \dots t_k)$ into a single TPTP formula, an additional axiom, corresponding to $\text{o\#f}() :- \text{o\#f}(t_1 \dots t_k)$, would need to be added to be able to derive $\text{o\#f}()$ from $\text{o\#f}(t_1 \dots t_k)$.

There are two major steps in this phase: flattening nested psoa formulas and splitting flat composite psoa formulas. For the first step, any atomic formula with nested psoa terms (which must be anonymous [\[1\]](#)) will be flattened: The original formula is replaced by a conjunction containing equations pairing fresh variables with the nested psoa terms and containing the atomic formula in which each nested psoa term is replaced by its corresponding variable. Flattening will be applied recursively to equations that contain psoa terms with nested psoa terms until all the psoa terms are flat.

The second step is only needed for flattened psoa formulas that apply a predicate (not for psoa terms that apply a function), where the OID, slot names and values, and tuple components are all constants or variables. The definition of the truth value of a psoa formula in [\[1\]](#) introduces splitting semantically as follows (the meaning of a psoa formula is *defined* via its elementary constructs):

$$\begin{aligned}
 & - TVal_{\mathcal{I}}(\text{o\#f}([\mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1}] \dots [\mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m}] \mathbf{p}_1 \rightarrow \mathbf{v}_1 \dots \mathbf{p}_k \rightarrow \mathbf{v}_k)) = \mathbf{true} \\
 & \text{if and only if} \\
 & TVal_{\mathcal{I}}(\text{o\#f}) = \\
 & TVal_{\mathcal{I}}(\text{o\#Top}([\mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1}])) = \dots = TVal_{\mathcal{I}}(\text{o\#Top}([\mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m}])) = \\
 & TVal_{\mathcal{I}}(\text{o\#Top}(\mathbf{p}_1 \rightarrow \mathbf{v}_1)) = \dots = TVal_{\mathcal{I}}(\text{o\#Top}(\mathbf{p}_k \rightarrow \mathbf{v}_k)) = \mathbf{true}.
 \end{aligned}$$

The composite psoa formula is split into a conjunction of $1+m+k$ subformulas, including 1 class membership formula, m single-tuple formulas and k (RDF-triple-like) single-slot formulas. Normalization performs such splitting syntactically.

Translation of Elementary PSOA RuleML Constructs. We define the translation function $\tau_{psoa}(\dots)$ mapping each PSOA/PS elementary construct to a TPTP construct as follows:

– Constants

In PSOA RuleML, constants have the form "literal"^^symspace, where `literal` is a sequence of Unicode characters and `symspace` is an identifier for a symbol space. There are also six kinds of shortcuts for constants, as shown in the production of `CONSTSHORT` [9]:

```
CONSTSHORT ::= ANGLEBRACKIRI
             | CURIE
             | ''' UNICODESTRING '''
             | NumericLiteral
             | '_' NCName
             | ''' UNICODESTRING ''' '@' langtag
```

In TPTP, a constant can be either an identifier starting with a lower-case letter, a single-quoted string or a numeric constant. In the current version of the translator, we translate constants of the form `'_' NCName` into a TPTP identifier by removing `'_'`, and the first character of `NCName` is converted to lower case. Constants of type `NumericLiteral` are kept without any change. In future development we may consider using single-quoted full URIs for all constants as a configuration option.

– Variables

A PSOA/PS variable is a '?'-preceded Unicode string. To translate it into a TPTP variable starting with an upper-case letter, we replace '?' with 'Q'. For example, a PSOA variable `?job` is mapped to a TPTP variable `Qjob`.

– Tuple Terms

A tuple term in PSOA/PS is of the form `o#Top(t1...tk)`. It associates the tuple `[t1...tk]` with the OID `o` (other tuple terms can use the same OID). To translate it into TPTP, we use a reserved predicate, `tupterm`, and use `o` as the first argument of the predicate. The `k` components of the tuple follow as the sequence of remaining arguments. Since `o#Top` is true for every `o`, `Top` is omitted without affecting the semantics. The result is a $(1+k)$ -ary term, `tupterm($\tau_{psoa}(o), \tau_{psoa}(t_1) \dots \tau_{psoa}(t_k)$)`. Note that the predicate name `tupterm` is polyadic – i.e., representing predicates of different arities – as allowed by the TPTP syntax.

– Slot Terms

A slot term in PSOA/PS has the form `o#Top(pi->vi)`. Its meaning is that the object with an OID `o` has a property `pi` and the property value is `vi`. We use another reserved predicate, `sloterm`, to represent this relationship in TPTP. `Top` is omitted as for tuple terms. The result is a ternary term, `sloterm($\tau_{psoa}(o), \tau_{psoa}(p_i), \tau_{psoa}(v_i)$)`, corresponding to an RDF triple.

– Membership Terms

Class membership terms in PSOA/PS are of the form `o#f()` (abridged `o#f`), meaning `o` is an instance of class `f`. In the translation, we use a third reserved predicate, `member`, so that the result is a binary term `member($\tau_{psoa}(o), \tau_{psoa}(f)$)`. In future versions, we may optionally use an alternative translation where `o#f()` would be translated as `f(o)`, i.e., treating

the class **f** as a unary predicate, for compatibility with other sources of TPTP formulas, such as the translator from RIF to TPTP⁹

– **Subclass Formulas**

Subclass formulas **c1 ## c2** in PSOA/PS are reused unchanged from RIF, meaning all the instances of class **c1** are also instances of class **c2**. To translate the subclass formula, a fourth reserved predicate **subclass** is used to represent the subsumption relation **##** in TPTP. The translation of the formula in TPTP is **subclass**($\tau_{psoa}(c1)$, $\tau_{psoa}(c2)$). Note that solely with such a translation, we are not able to infer the inheritance **o # c2** just from the translation of **o # c1** and **c1 ## c2**. In order to make that inference, this extra inference axiom for inheritance is needed in TPTP:

$$! [X, Y, Z] \text{ (member}(X, Y) \ \& \ \text{subclass}(Y, Z) \Rightarrow \text{member}(X, Z))$$

– **Equality Formulas**

In PSOA/PS an equality formula **a = b** means the terms **a** and **b** are equal. This formula can be translated to $\tau_{psoa}(a) = \tau_{psoa}(b)$ in TPTP.

– **Rule Implications**

In PSOA/PS a rule is represented by $\varphi :- \psi$, meaning formula φ is implied by formula ψ . It can be translated to $\tau_{psoa}(\psi) \Rightarrow \tau_{psoa}(\varphi)$.

Table 2. Mapping from PSOA/PS constructs to TPTP constructs

PSOA/PS Constructs	TPTP Constructs
o # Top ($t_1 \dots t_k$)	tupterm ($\tau_{psoa}(o), \tau_{psoa}(t_1) \dots \tau_{psoa}(t_k)$)
o # Top (p -> v)	sloterm ($\tau_{psoa}(o), \tau_{psoa}(p), \tau_{psoa}(v)$)
o # f ()	member ($\tau_{psoa}(o), \tau_{psoa}(f)$)
a ## b	subclass ($\tau_{psoa}(a), \tau_{psoa}(b)$)
a = b	$\tau_{psoa}(a) = \tau_{psoa}(b)$
AND ($f_1 \dots f_n$)	($\tau_{psoa}(f_1) \ \& \ \dots \ \& \ \tau_{psoa}(f_n)$)
$\varphi :- \psi$	$\tau_{psoa}(\psi) \Rightarrow \tau_{psoa}(\varphi)$

Table 2 summarizes the mapping from elementary PSOA/PS constructs to TPTP constructs, including one extra row for conjunctions, as needed in rule premises. The mapping is sufficient for the translation of a KB but not yet for a query: we expect to get the bindings for any query variables. Since query answering in VampirePrime is done through answer predicates, we introduce a reserved predicate **ans** as the answer predicate and map a PSOA query **q** into the following formula

$$! [X1, X2, \dots] \text{ (} \tau_{psoa}(q) \Rightarrow \text{ans}("?X1 = ", \tau_{psoa}(X1), "?X2 = ", \tau_{psoa}(X2), \dots))$$

where $X1, X2, \dots$ are free variables in **q**. Answers from VampirePrime are of the form

⁹ http://riazanov.webs.com/RIF_BLD_to_TPTP.tgz

```
ans("?X1 = ", v1, "?X2 = ", v2, ...)
```

where $v1, v2, \dots$ are bindings for the variables. When the query has no variables, `ans` is used alone as the conclusion. A sample will be given in the next section.

5.2 Translator Implementation

In the current version of PSOA2TPTP, we have implemented the translation for a PSOA RuleML subset where the accepted constants are numerals and short-form RIF-like local constants starting with ‘_’. Following is a sample translation, where the two conjunctions resulting from normalization are implicit in the enclosing `Group`:

– **Input KB:**

```
Document(
  Group(
    _f1#_family(_Mike _Jessie _child->_Fred _child->_Jane)
    _Amy#_person([_female] [_bcs _mcs _phd] _job->_engineer)
  )
)
```

– **Normalized KB:**

```
Document(
  Group(
    _f1#_family() _f1#Top(_Mike _Jessie)
    _f1#Top(_child->_Fred) _f1#Top(_child->_Jane)
    _Amy#_person() & _Amy#Top(_female)
    _Amy#Top(_bcs _mcs _phd) _Amy#Top(_job->_engineer)
  )
)
```

– **Query:** `_Amy#_person(_job->?job)`

– **Translator Output:**

```
fof(ax1, axiom,
  member(f1, family) & tupterm(f1, mike, jessie) &
  sloterm(f1, child, fred) & sloterm(f1, child, jane)).
fof(ax2, axiom,
  member(amy, person) & tupterm(amy, female) &
  tupterm(amy, bcs, mcs, phd) & sloterm(amy, job, engineer)).
fof(query, theorem,
  ![Qjob]: ((member(amy, person) & sloterm(amy, job, Qjob))
    => ans("?job = ", Qjob) )).
```

– **VampirePrime Output:**

```
Proof found.
...
... | «ans»("?job = ", engineer) ...
```

The sample KB has two *psoa* formulas as facts. The first fact has one tuple for the family’s adults, where `_Mike _Jessie` is equivalent to `[_Mike _Jessie]`, a shortcut allowed only in single-tuple *psoa* terms; it has two slots for the family’s children. The second fact has two tuples, of lengths 1 and 3, and also a slot. The two formulas in the first stage are broken into two conjunctions of elementary constructs, as shown in the normalized KB above. In the second stage, each construct is mapped to a corresponding TPTP term. The above translator output contains three translations, of which the first two are for the KB and the last is for the query.¹⁰ In the VampirePrime output, `«ans»(" ?job = ",engineer)` indicates one binding, `engineer`, is obtained for the variable `?job`.

6 RESTful Web API

Representational State Transfer (REST) is an architectural style for distributed systems, specified in [11]. A RESTful Web API is an API implemented by using HTTP (operations, URIs, Internet media types, response codes) and by conforming to the *architectural constraints* specified in REST. We have implemented a RESTful Web API consisting of two resources, see URIs in Table 3: a resource representing the PSOA2TPTP translation component, and a resource representing the VampirePrime reasoner component. As shown in Table 4, the HTTP POST method is allowed and the *application/json* Internet media type is supported for the listed resources.

For example, to translate PSOA RuleML into TPTP, an HTTP POST¹¹ request with a JSON-encoded PSOA document in the body is sent to the *Translate* resource URI. The response will be a JSON encoding of the translated TPTP document.

Table 3. RESTful resource URIs

Resource	URI
Translate	<code>http://example.ws/translate</code>
Execute	<code>http://example.ws/execute</code>

To execute the translated TPTP sentences in the VampirePrime reasoner, the *Execute* resource mentioned in Table 4 should be used by sending an HTTP POST request containing an *application/json* encoding of the TPTP sentences to the *Execute* resource URI. The server will then return the raw output stream (*text/plain*) of the VampirePrime-generated solutions (see listing 1.5 in [8]).

¹⁰ PSOA2TPTP targets VampirePrime by using TPTP’s *axiom* for the KB and *theorem* for queries, while [10] would suggest the use of *conjecture* for queries.

¹¹ POST is chosen over PUT to reflect idempotency of the translation service: a new translation (instance) is produced for each request.

Table 4. RESTful resources

Resource	Methods	Description
Translate	POST	This resource represents the PSOA2TPTP translator. <i>Media types: application/json</i>
Execute	POST	This resource represents the VampirePrime reasoner. <i>Media types: application/json, text/plain</i>

7 Conclusion

To enable rule/theorem prover interoperability, we implemented a first version of the PSOA2TPTP translator. It takes a document in PSOA/PS as input and generates a semantics-preserving TPTP document. Through the translator, PSOA RuleML documents are translated into the TPTP format and can then be executed by the VampirePrime reasoner or other TPTP systems.

Our work makes heavy use of the ANTLR v3 parser generator framework. We (1) rewrite the complete PSOA/PS EBNF grammar into an $LL(1)$ grammar; (2) construct an intermediate AST using ANTLR's tree rewrite mechanisms embedded in the parser grammar; (3) develop reusable tree grammars for parsing the AST; and (4) embed code into the tree grammars to generate ASOs and TPTP documents.

Future work on the PSOA RuleML implementation includes: (1) Extending the PSOA2TPTP translator capability to handle all PSOA RuleML constructs, introducing another reserved predicate for functional psOA terms; (2) implementing the fully-ASO-based translator; (3) creating a testbed for rigorously testing PSOA RuleML implementations; (4) 'inverting' PSOA2TPTP into a TPTP2PSOA translation for the Horn subset of first-order logic; (5) deploying PSOA2TPTP in the Clinical Intelligence use case [12], where PSOA rules are used to define a semantic mapping for a hospital data warehouse.

References

1. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: Bassiliades, N., et al. (eds.) RuleML 2011 - Europe. LNCS, vol. 6826, pp. 194–211. Springer, Heidelberg (2011)
2. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. *Journal of Emerging Technologies in Web Intelligence* 4(2), 343–353 (2010), <http://ojs.academypublisher.com/index.php/jetwi/article/view/0204343353>
3. Boley, H., Kifer, M.: A Guide to the Basic Logic Dialect for Rule Interchange on the Web. *IEEE Transactions on Knowledge and Data Engineering* 22(11), 1593–1608 (2010)
4. Riazanov, A., Voronkov, A.: The Design and Implementation of Vampire. *AI Communications* 15(2-3), 91–110 (2002)
5. Riazanov, A., Aragao, M.A.: Incremental Query Rewriting with Resolution. In: *Canadian Semantic Web II* (2010)

6. Al Manir, M.S., Riazanov, A., Boley, H., Baker, C.J.O.: PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes. In: Bikakis, A., Giurca, A. (eds.) RuleML 2012. LNCS, vol. 7438, pp. 280–288. Springer, Heidelberg (2012)
7. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D. (eds.): Compilers: Principles, Techniques, and Tools, 2nd edn. Pearson/Addison Wesley, Boston (2007)
8. Zou, G., Peter-Paul, R.: PSOA2TPTP: Designing and Prototyping a Translator from PSOA RuleML to TPTP Format. Technical report, http://psoa2tptp.googlecode.com/files/PSOA2TPTP_Report_v1.0.pdf
9. Polleres, A., Boley, H., Kifer, M.: RIF Datatypes and Built-ins 1.0 (June 2010), W3C Recommendation, <http://www.w3.org/TR/rif-dtb>
10. Sutcliffe, G.: The TPTP Problem Library, <http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml>
11. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, Irvine, California, USA (2000)
12. Riazanov, A., Rose, G.W., Klein, A., Forster, A.J., Baker, C.J.O., Shaban-Nejad, A., Buckeridge, D.L.: Towards Clinical Intelligence with SADI Semantic Web Services: a Case Study with Hospital-Acquired Infections Data. In: Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences, SWAT4LS 2011, pp. 106–113. ACM, New York (2012)

PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes

Mohammad Sadnan Al Manir¹, Alexandre Riazanov¹,
Harold Boley², and Christopher J.O. Baker¹

¹ Department of Computer Science and Applied Statistics
University of New Brunswick, Saint John, Canada
{sadnan.almanir,bakerc}@unb.ca, alexandre.riazanov@gmail.com

² Information and Communications Technologies
National Research Council Canada
harold.boleynrc.gc.ca

Abstract. PSOA RuleML is a rule language which introduces positional-slotted, object-applicative terms in generalized rules, permitting relation applications with optional object identifiers and positional or slotted arguments. This paper describes an open-source PSOA RuleML API, whose functionality facilitates factory-based syntactic object creation and manipulation. The API parses an XML-based concrete syntax of PSOA RuleML, creates abstract syntax objects, and uses these objects for translation into a RIF-like presentation syntax. The availability of such an API will benefit PSOA rule-based research and applications.

1 Introduction

F-logic [1] and W3C RIF [2] define objects (frames) separately from functions and predicates. POSL and PSOA RuleML [3,4] provide an integration of object identifiers with applications of functions or predicates to positional or slotted arguments, called **positional-slotted, object-applicative** (psoa) term. While RIF requires different kinds of terms for positional and slotted information as well as for frames and class memberships, PSOA RuleML can express them with a single kind of psoa term. As a result, PSOA rules permit a compact way of authoring rule bases, which are as expressive as POSL and semantically defined in the style of RIF-BLD. The constructs of PSOA RuleML are described in [3] in detail. In this paper, ‘psoa’ in lower-case letters refers to a kind of terms while ‘PSOA’ in upper-case letters refers to the language.

Here we describe an open-source PSOA RuleML API. The inspiration comes from well-known APIs for Semantic Web languages such as the OWL API [5] and the Jena API [6]. The existence of these APIs facilitates a lot of experimental research and development in Semantic Web technologies and we hope that our API will have a similar effect on the PSOA adoption. Our API allows creation of objects corresponding to PSOA constructs, such as constants, variables, tuples, slots, atoms, formulas, rules, etc., using factory-based method calls, as well as traversal of those objects using simple recursive traversal. Moreover, it supports

parsing of XML-based PSOA documents and generation of presentation syntax. Thus, users will be able to employ this API for rule processing, including rule authoring, rule translation into other languages, rule-based applications, and rule engines.

Due to space constraints, here we briefly describe the key features of the PSOA RuleML presentation syntax. The language is best described using conditions and rules built over various terms, centered around *psoa* terms in particular.

We begin with the disjoint sets of alphabets of the language. The alphabets include a countably infinite set of constant and variable symbols, connective symbols (e.g., **And**, **Or**, **-**), quantifiers (e.g., **Exists**, **Forall**), and other auxiliary symbols (e.g., **=**, **#**, **##**, **->**, **External**, **Group**, **(**, **)**, **<**, **>**, **^^**, **-**).

The language contains literal constants and IRI constants, the latter sometimes abbreviated as short constants.

The following examples illustrate double-type and string-type literal constants:

```
"27.98"^^xs:double           "The New York Times"^^xs:string
```

Constants like **family**, **kid** are short constants.

Each variable name is preceded by a '?' sign, such as ?1, ?Hu, ?Wi, etc.

In a *psoa* term, the function or predicate symbol is instantiated by an object identifier (OID) and applied to zero or more positional or named arguments. The positional arguments are referred to as tuples while named arguments (attribute-value pairs) are called slots.

For example, a *psoa* term (an atom), containing **family** relation with the OID ?inst, tuples husband ?Hu, and wife ?Wi, along with a slot **child->**?Ch can be represented as follows:

```
?inst#family(?Hu ?Wi child->?Ch)
```

Terms include *psoa* terms as well as several different types of logic terms, such as constants and variables, equality, subclass, and external terms.

An atomic formula with **f** as the predicate is defined as **f(...)** in general. PSOA applies a syntactic transformation to incorporate the OID, which results in the objectified atomic formula **o#f(...)**, with **o** as the OID and **f** acting as its class. The OID is represented by a stand-alone '_' for a ground (variable-free) fact, an existentially-scoped variable for a non-ground fact or an atomic formula in a rule conclusion, and a stand-alone '?' as an anonymous variable for any other atomic formula.

Condition formulas are used as queries or rule premises. Conjunction and disjunction of formulas are denoted by **And** and **Or**, respectively. Formulas with existentially quantified variables are also condition formulas. An example of a condition formula is given below:

```
And(?2#married(?Hu ?Wi) Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch)))
```

Aside from the condition formulas, the premise can also contain atomic formulas and external formulas.

A conclusion contains a head or conjunction of heads. A head refers to an atomic formula which can also be existentially quantified. A conclusion example is given below:

```
Exists ?1 (?1#family(husb->?Hu wife->?Wi child->?Ch))
```

An implication contains both conclusion and condition formulas. A clause is either an atomic formula or an implication. A rule is generated by a clause within the scope of the `Forall` quantifier or solely by a clause. Several formulas can be collected into a `Group` formula.

The `Group` formula below contains a universally quantified formula, along with two facts. The `Forall` quantifier declares the original universal argument variables as well as the generated universal OID variables `?2`, `?3`, `?4`. The infix `:-` separates the conclusion from the premise, which derives the existential family frame from a `married` relation `And` from a `kid` of the `husb` `Or` `wife`. The following example from [3] shows an objectified form on the right.

<pre>Group (Forall ?Hu ?Wi ?Ch (family(husb->?Hu wife->?Wi child->?Ch) :- And(married(?Hu ?Wi) Or(kid(?Hu ?Ch) kid(?Wi ?Ch)))) married(Joe Sue) kid(Sue Pete)))</pre>	<pre>Group (Forall ?Hu ?Wi ?Ch ?2 ?3 ?4 (Exists ?1 (?1#family(husb->?Hu wife->?Wi child->?Ch) :- And(?2#married(?Hu ?Wi) Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch)))) _1#married(Joe Sue) _2#kid(Sue Pete)))</pre>
---	--

The objectified family term in the rule conclusion is slotted with 3 slots:

```
?1#family(husb->?Hu wife->?Wi child->?Ch)
```

The rules's condition formulas use the relations `married` and `kid`, containing only 2-tuples `Hu`, `Wi`, and `Ch`:

```
?2#married(?Hu ?Wi) ?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch)
```

The next section will describe the API components and their uses. We begin by describing the organization of the package, then illustrate the object creation and traversal as well as parsing the PSOA/XML input, and rendering in presentation syntax. For all of these operations, we use the objectified family example above. Finally, we conclude by mentioning the scope of using our API with other complementary tools and potential work directions in the future.

2 The API Structure and Functionality

2.1 Package Organization

The API is divided into two main components: one is for the creation and traversal of abstract syntax objects and the other is for parsing and rendering of those objects.

The *AbstractSyntax* is the top level class for factories and contains all Java interfaces for different types of abstract syntax objects. A simple implementation of *AbstractSyntax* interfaces is in the *DefaultAbstractSyntax* class, which is suitable for most purposes. However, more demanding uses may require custom implementations of the interfaces.

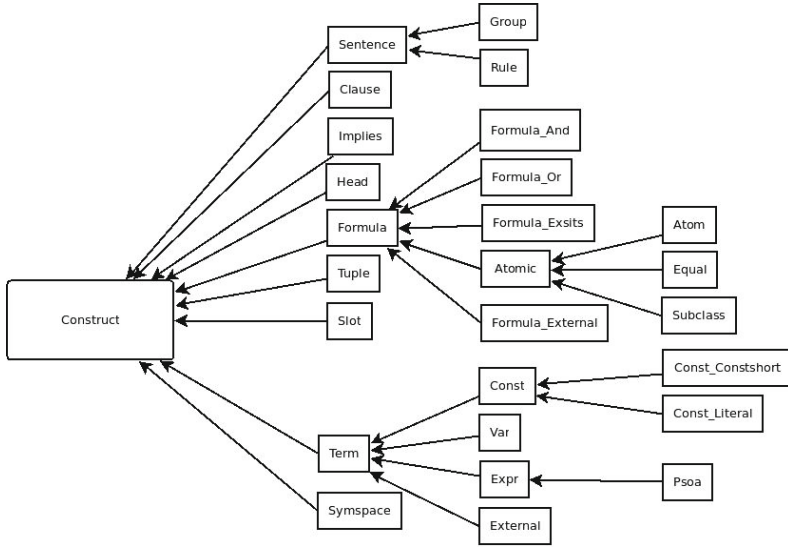


Fig. 1. The API Structure

The package also contains a *Parser* class, which provides PSOA/XML parsing and translation into presentation syntax. Parsing is implemented using the Java Architecture for XML Binding (JAXB) [7], which creates equivalent Java objects based on XML schema files. The schema is a straight-forward encoding of the syntactic construct hierarchy and is available in [8]. The parsed XML is then converted to the abstract syntax by calling factory methods.

Figure 1 presents the most important classes and interfaces implementing different types of syntactic constructs. Each of the names in a rectangular box represents a Java interface, and is kept as close as possible to the presentation syntax construct names. The corresponding implementations of these interfaces use Java inheritance, shown by the solid arrows.

The interface **Construct** sits at the top of the hierarchy. **Group** can be populated with more **Groups** and **Rules**. Both universal facts and universal rules are represented by the **Rule** class, which encapsulates a **Clause**. The interface **Clause** represents either an implication or an atomic formula. Implication is represented by the interface **Implies** whereas **Atomic** represents atomic formulas. The generalized interface **Atomic** is implemented either by **Atom** (representing a psOA term like `?2#married(?Hu ?Wi)`) or by **Subclass** (representing a subclass term like `student##person`) or by **Equal** (equality term like `?cost = ?spent`).

Implementations of the generalized **Formula** interface represent either disjunction, conjunction, or existential formulas by implementing **Formula_Or** (represented as `Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch))`), **Formula_And** (`And` instead of `Or`), and **Formula_Exists**, respectively. In addition to atomic formulas, external formulas (**External**(`func:numeric-add(?cost1 ?cost2)`)) can be represented using the **Formula_External** interface. The generalized interface **Term** is

represented by implementing `Const` for different kinds of constants, `Var` for variables like `?Hu`, `?Wi`, `Expr` for expressions denoting `psoa` terms, and `External` for external Expressions. Constants can be either `Const_Literal` (e.g., `"47.5"^^xs:float`, with `Symspace` referring to `xs:float`) or `Const_Constshort` (e.g., `family ,kid`). Finally, the interface `Psoa` is implemented to represent objectified functions or predicates with membership symbol `#` and tuples and slots as arguments, e.g., `inst#family(Homer Merge child->Bart)`. The internal nodes `Sentence`, `Formula`, `Atomic`, `Term`, `Const`, and `Expr`, are generalized classes and implemented by more specific classes.

2.2 Construction of Abstract Syntax Objects

The abstract syntax objects are constructed by factory-based *createX* methods calls, *X* being the object type name. The rest of this paper represents each method in *emphasized* font. A factory can be created as follows:

```
DefaultAbstractSyntax absSynFactory = new DefaultAbstractSyntax()
```

We are going to illustrate the creation of facts and rules below.

Construction of Facts. A fact is of type `Atomic`. Let us look at the first fact that tells us Joe and Sue are married to each other with the OID `_1`, whereas the second fact says Pete is the kid of Sue with the OID `_2`, each fact referring to a `psoa` term.

The creation of fact `_1#married(Joe Sue)` starts by creating the four constants `_1`, `married`, `Joe`, `Sue` as `const_1`, `const_married`, `const_Joe`, `const_Sue`, respectively using the method *createConst_Constshort*.

```
Const_Constshort const_1 = absSynFactory.createConst_Constshort("_1")
Const_Constshort const_married = absSynFactory
    .createConst_Constshort("married")
Const_Constshort const_Joe = absSynFactory
    .createConst_Constshort("Joe")
Const_Constshort const_Sue = absSynFactory
    .createConst_Constshort("Sue")
```

Tuples `const_Joe` and `const_Sue` are constructed by the method *createTuple*. The list of such tuples is referred to as a `tuplesList`.

```
Tuple tuples = absSynFactory.createTuple(tuplesList)
```

Method *createPsoa* assembles `_1`, `married` and `tuples` into a `psoaTerm`, while *null* indicates the absence of slots.

```
Psoa psoaTerm = absSynFactory
    .createPsoa(const_1, const_married, tuples, null)
```

Here is how we create an atom:

```
Atom atom = absSynFactory.createAtom(psoaTerm)
```

Thus, we use the method *createAtom* for creating a fact of type `Atom`, *createEqual* for a fact of type `Equal`, and *createClass* for type `Subclass`. This creation is completed by the *createClause* and *createRule* method calls. The representation for creating the fact `_2#kid(Sue Pete)` is similar to the method calls described above, hence omitted.

Construction of Rules. A rule contains condition and conclusion. We will start with the condition formula, which is a conjunction of the atomic formula `?2#married(?Hu ?Wi)` and disjunction of two atomic formulas, `?3#kid(?Hu ?Ch)` and `?4#kid(?Wi ?Ch)`.

```
Forall ?Hu ?Wi ?Ch ?2 ?3 ?4 (
  Exists ?1 (
    ?1#family(husb->?Hu wife->?Wi child->?Ch) :-
      And(?2#married(?Hu ?Wi) Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch)))
  )
)
```

The following code snippet creates the disjunction of two atoms. Method *createFormula_Or* defines the disjunction of two atomic formulas, `atomOr_1` and `atomOr_2`. The OIDs `?3` and `?4` (`var_4`), as well as tuples `?Hu` (`var_Hu`), `?Wi` (`var_Wi`) and `?Ch` (`var_Ch`), are variables. Only the construction of `?3` (`var_3`) is shown below to avoid repetition (...).

```
Var var_3 = absSynFactory.createVar("3")
...
Tuple tuples = absSynFactory.createTuple(tuplesList_1)
Psoa psoaTerm_1 = absSynFactory
  .createPsoa(var_3, const_kid, tuples, null)
Atom atomOr_1 = absSynFactory.createAtom(psoaTerm_1)
...
Atom atomOr_2 = absSynFactory.createAtom(psoaTerm_2)
```

Both of the atomic formulas `atomOr_1` and `atomOr_2` are in a list called `formulaOrList`.

```
Formula_Or formula_Or = absSynFactory.createFormula_Or(formulaOrList)
```

The conjunction of the newly created `formula_Or` and another atomic formula `atom_And`, `?2#married(?Hu ?Wi)` is described next. Here `var_2`, `var_Hu` and `var_Wi` denote the variables `?2`, `?Hu` and `?Wi`, respectively. The code below does this using the method *createFormula_And*. The list `formulaAndList` contains atomic formulas `atom_And` and `formula_Or`. The conjunction formula `formula_And` is the rule premise and created as follows:

```
Formula_And formula_And = absSynFactory
  .createFormula_And(formulaAndList)
```

We now move on to the rule Head creation, which is a `psoa` term containing the OID `?1` with family class name and three slots, `husb->?Hu`, `wife->?Wi`, and `child->?Ch`, in `?1#family(husb->?Hu wife->?Wi child->?Ch)` as arguments. These slots will be called `slot_1`, `slot_2`, and `slot_3`, respectively.

```
Var var_1 = absSynFactory.createVar("1")
...
Slot slot_1 = absSynFactory.createSlot(const_husb, var_Hu)
Slot slot_2 = absSynFactory.createSlot(const_wife, var_Wi)
Slot slot_3 = absSynFactory.createSlot(const_child, var_Ch)
```

The list of slots `slot_1`, `slot_2` and `slot_3`, called `slotsList`, is used to create the `psoa` term.

```
Psoa psoa = absSynFactory.createPsoa(var_1, const_family, null,slotsList)
```

The atom created is `atom_head`, and thus the rule head is created by the method `createHead`, where the variable `var_1` is existentially quantified. Both existentially and universally quantified variables are treated as a list of variables, called `varsList`.

```
Head rule_head = absSynFactory.createHead(varsList, atom_head)
```

The method `createImplies` combines the rule head, `rule_head` and the rule premise, `formula_And`, into an implication. Method `createClause` creates the implication. Finally, method `createRule` collects all the universally quantified variables, `var_Hu`, `var_Wi`, `var_Ch`, `var_2`, `var_3`, and `var_4` into a `varsList` and creates the rule with the clause.

```
Implies implication = absSynFactory.createImplies(rule_head, formula_And)
Clause clause = absSynFactory.createClause(implication)
Rule rule = absSynFactory.createRule(varsList, clause)
```

2.3 Abstract Syntax Structure Traversal

Our implementation recursively traverses the object tree generated from the abstract syntax structure and is usually simpler than writing visit methods [9] as used in OWL API.

All components of the abstract syntax structure can be accessed directly by the corresponding accessor methods, which are *getX* methods. The generalized classes (see Figure II) are `Sentence`, `Formula`, `Atomic`, `Term`, `Const`, and `Expr`, containing *isX* methods to recognize the specific instance types. Alternatively, specific classes of particular instances have to be identified, e.g., by using `instanceof`.

For an atomic formula, an *isX* method in `Atomic` class needs to recognize if the instance is of type `Atom`, `Subclass`, or `Equal` object. This principle applies to each of the generalized classes.

For example, *isEqual* method in generalized class `Atomic` recognizes the instance of `Equality` atom `?cost = "47.5"^^xs:float`. Immediately, a cast is made as the instance type `Equal` and *getLeft* and *getRight* methods are called, each referring to an instance of another generalized class `Term`. Class `Term` contains appropriate *isX* methods, which use similar techniques to find out if the instance is of type `Const`, or `Var`, or an `External` expression.

```
assert this instanceof AbstractSyntax.Equal
return (AbstractSyntax.Equal) this
...
AbstractSyntax.Term getLeft()
AbstractSyntax.Term getRight()
```

Method *getLeft*, in this case, retrieves the instance of `Var` and thus string variable `?cost` is retrieved by the method *getName* as the variable instance. On the other hand, *getRight* refers to the instance of type `Const`. Method *isConstLiteral* recognizes `ConstLiteral` involving the literal and the type `float` involving the instance of type `SymSpace`. The literal object `47.5` is retrieved as string by the method *getLiteral*. Finally, `xs:float` object is retrieved by the method *getValue* as an instance of type `SymSpace`.

Thus, the traversal of objects in the API structure follows the same strategy of going down to most specific instances in a recursive manner for both facts and rules.

2.4 Parsing and Rendering

Aside from creating and traversing objects, the API is able to parse PSOA/XML inputs and render them in human readable presentation syntax.

In section 2.1 we discuss an XML schema for PSOA RuleML. We generate the XML parser with the help of JAXB, which creates Java classes from a schema traversal, where the ultimate output of the parser is abstract syntax objects.

The following example shows a transformation of an XML input for a fact and its rendering in presentation syntax.

```

<Atom>
  inst1#family(Joe Sue Child->Pete)
  <Member>
    <instance>
      <Const type="\&psoa;iri">inst1</Const>
    </instance>
    <class>
      <Const type="\&psoa;iri">family</Const>
    </class>
  </Member>
  <tuple>
    <Const type="\&psoa;iri">Joe</Const>
    <Const type="\&psoa;iri">Sue</Const>
  </tuple>
  <slot>
    <Const type="\&psoa;iri">Child</Const>
    <Const type="\&psoa;iri">Pete</Const>
  </slot>
</Atom>

```

A *toString* method in each class implements this pretty-printing, which follows the same traversal procedure described in section 2.3.

3 Conclusion and Future Work

The API is open-source and hosted in [10]. The companion effort PSOA2TPTP [11] has developed a reference translator for PSOA RuleML, which facilitates inferencing using TPTP reasoners (see e.g., [12]). One component of the translator is a parser for the presentation syntax. Our API will greatly benefit from including this presentation syntax parser. The other component of the PSOA2TPTP translator is its mapping from abstract syntax objects to TPTP. Combined with our API, this will also make PSOA/XML executable on the TPTP-aware VampirePrime [13] reasoner.

Currently, the API can render PSOA/XML only into presentation syntax. As an extension, we plan to also include the translation of abstract syntax objects back to PSOA/XML.

We have been using the API in our HAIKU work [14], where PSOA is used to capture semantic modeling of relational data and needed, at least, to support

authoring, including syntactic and, to some extent, logical validation (consistency checking). We also plan to use it for automatic generation of Semantic Web services from declarative descriptions.

PSOA RuleML API has become an input to the Object Management Group's API4KB effort [15], which tries to create a universal API for knowledge bases that among other things combines the querying of RDF-style resource descriptions, ODM/OWL2-style ontologies, and RIF RuleML-style rules.

References

1. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. *J. ACM* 42(4), 741–843 (1995)
2. Boley, H., Kifer, M.: A Guide to the Basic Logic Dialect for Rule Interchange on the Web. *IEEE Trans. Knowl. Data Eng.* 22(11), 1593–1608 (2010)
3. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: Bassiliades, N., et al. (eds.) *RuleML 2011 - Europe*. LNCS, vol. 6826, pp. 194–211. Springer, Heidelberg (2011), <http://dblp.uni-trier.de/db/conf/ruleml/ruleml2011e.html#Boley11>
4. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. *Journal of Emerging Technologies in Web Intelligence* 4(2), 343–353 (2010), <http://ojs.academpublisher.com/index.php/jetwi/article/view/0204343353>
5. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL Ontologies. *Semantic Web* 2(1), 11–21 (2011)
6. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. Technical Report HPL-2003-146, Hewlett Packard Laboratories (2003)
7. Fialli, J., Vajjhala, S.: Java Architecture for XML Binding (JAXB) 2.0. Java Specification Request (JSR), 222 (October 2005)
8. <http://code.google.com/p/psoa-ruleml-api/source/browse/trunk/PSOARuleML-API/src/main/resources/>
9. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.M.: Design Patterns: Abstraction and Reuse of Object-Oriented Design. In: Wang, J. (ed.) *ECOOP 1993*. LNCS, vol. 707, pp. 406–431. Springer, Heidelberg (1993)
10. PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes (2012), <http://code.google.com/p/psoa-ruleml-api/>
11. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners. In: Bikakis, A., Giurca, A. (eds.) *RuleML 2012*. LNCS, vol. 7438, pp. 264–279. Springer, Heidelberg (2012)
12. System on TPTP, <http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP>
13. VampirePrime Reasoner, <http://riazanov.webs.com/software.html>
14. Riazanov, A., Rose, G.W., Klein, A., Forster, A.J., Baker, C.J.O., Shaban-Nejad, A., Buckeridge, D.L.: Towards Clinical Intelligence with SADI Semantic Web Services: A Case Study with Hospital-Acquired Infections Data. In: Paschke, A., Burger, A., Romano, P., Marshall, M.S., Splendiani, A. (eds.) *SWAT4LS*, pp. 106–113. ACM (2011)
15. <http://www.omgwiki.org/API4KB/lib/exe/fetch.php?media=api4kb:rfp.pdf>

Syntax Reuse: XSLT as a Metalanguage for Knowledge Representation Languages

Tara Athan

Athan Services, Ukiah, CA, USA
taraathan@gmail.com

Abstract. We present here MXSL, a subset of XSLT re-interpreted as a syntactic metalanguage for RuleML with operational semantics based on XSLT processing. This metalanguage increases the expressivity of RuleML knowledge bases and queries, with syntactic access to the complete XML tree through the XPath Data Model. The metalanguage is developed in an abstract manner, as a paradigm applicable to other KR languages, in XML or in other formats.

Keywords: Metalanguage, metamodel, RuleML, operational semantics, XSLT.

1 Introduction

One of the most important issues in the field of knowledge representation (KR) is the issue of semantic interoperability. This issue might be described as the compounding of information overload with representation-format overload. Representation formats abound because information collections (datasets, knowledge bases, metadata, ...) have a wide diversity of characteristics (numerical, textual, hierarchical, ...) and uses (quantitative processing, search, reasoning, ...), and diverse formats have been developed to meet these diverse needs.

One of the most widely-used formats is XML, due in part to its capability at representing both structured data and document markup, as well as the capability to specialize the syntax through schema languages, including XSD [1], Relax NG [2] and Schematron [3]. XML has been designated the foundational syntax of the semantic web [4], although other formats, such as JSON [5] may play a larger role in the future.

The need for compatibility between semantic and syntactic models of structured XML has been addressed from an RDF perspective in [6] and [7]. In the latter study, the syntactic XPath Data Model and the semantic RDFS model, are considered in parallel, as being alternate, and partially compatible, models of the same subject, an XML instance. This issue was considered again from the perspective of an XML-based Common Logic syntax in [8].

In this paper, we consider the case where the XML instance is itself a semantic representation, in particular instances of the RuleML [9] language, possibly containing embedded structured data. We investigate the use of a syntactic model of XML to describe correspondences between syntactic substructures of RuleML.

The goal of this investigation is to develop a syntactic metalanguage for RuleML in order to

1. enable reasoning over large, structured legacy datasets (e.g. in XML or flat files) without large-scale conversion of compact representations into the more verbose RuleML format;
2. allow user-specified deductive systems for reasoning over RuleML knowledge bases via axiom schemas;
3. enhance the RuleML query capability.

In addition, the principles of this metalanguage development are abstracted so that it may be viewed as a process that can be applied to the development of metalanguages for other (XML and non-XML) KR formats.

We define an operational semantics of the metalanguage as a "metainterpretation" transformation. This transformation maps the syntactic representations in the metalanguage into syntactic representations in the base language by substitution of (lexical) values for the metavariables, which we will call parameters to distinguish them from the logical variables.

Thus, a full interpretation of an instance of the metalanguage consists of two distinct phases;

1. the application of the metainterpretation to transform the metalanguage instance to another instance that is purely in the base language;
2. the application of the base language's model-theoretic interpretation to transform the derived instance to truth values.

The proof theory of the metalanguage is defined similarly. The deductive apparatus of the base language is extended to include deduction from metalanguage axioms to base language axioms obtained from metainterpretation with a suitable binding of the parameters.

The obvious model for a transformation language on XML-based metalanguage input is XSLT [10]. XSLT is a declarative language based on the same abstract data model underlying XPath and XQuery [11].

XSLT is a very rich language, which is in fact Turing complete [12]. We require only a small subset of the capabilities of XSLT for our purposes. In particular, our metalanguage should have the capability to

- add an element node to the output tree, given the local name and namespace of the element as parameters;
- add an attribute node to a particular element node, given the local name, namespace and value as parameters;
- add a text node as the child of an element;
- iteratively add, in a specified order, a number of nodes to the output tree, based on a set of parameter bindings represented in structured XML;
- express the hierarchical and sequential structure of the output tree isomorphically according to the corresponding structure of the metalanguage instance.

In addition, we have the following operational requirements: Instances in the meta-language, which we will call MXSL, will:

- validate against the XSLT schema,
- be self-contained, and
- be self-executing.

2 MXSL Syntax

The core MXSL syntax is a subset of XSLT 2.0 syntax which uses primarily the following elements:

- `<xsl:for-each>`, for iteration
- `<xsl:element>`, for construction of XML elements
- `<xsl:attribute>`, for addition of attributes to XML elements
- `<xsl:copy-of>` with `@select`, for generation of the values of attributes and contents of element

Additional elements which are allowed in MXSL in a restricted form (as indicated in parentheses) are:

- `<xsl:stylesheet>` (once as the root element)
- `<xsl:variable>` (once as a header to define the parameter bindings and also as the first children of `<xsl:for-each>`, once for each parameter in the iteration)
- `<xsl:template match="/">` (once for the MXSL body)
- `<xsl:processing-instruction>` (zero to many times as the first children of `<xsl:template>`)
- `<ruleml:RuleML>` (once as the last child of `<xsl:template>` to define the root for the output document)
- `<xsl:comment>` (arbitrarily within the `<ruleml:RuleML>` element)

A Relax NG schema that may be used to verify that an MXSL instance does not exceed these restrictions is given at `rnc/mxsl.rnc`.¹

3 Example: Captured Strings

This example illustrates the definition of a function that takes as argument a character sequence, with some restrictions, and produces a name based on that string, equivalent to the captured string syntax of IKL [13]. The capability introduced by such an axiom schema is extremely useful, for example, in setting naming conventions that identify types, or constructing meaningful object identifiers calculated from object properties.

¹ The path to the directory for all files referenced in this paper is <http://athant.com/mxsl>.

A sample RuleML axiom that will be used to generate the axiom schema is

```
<ruleml:Equal>
  <ruleml:Expr>
    <ruleml:Fun
      iri="http://athant.com/mxsl/vocab#constant"/>
      <ruleml:Data xsi:type="xs:string">stringParam</Data>
    </ruleml:Expr>
  <ruleml:Ind>stringParam</ruleml:Ind>
</ruleml:Equal>
```

RuleML syntax does not allow us to generalize the character sequence `stringParam` that appears as the simple content of both the `<ruleml:Data>` and `<ruleml:Ind>` elements. That is, we can replace the `<ruleml:Data>` data constant with a variable `<Var>` and the `<ruleml:Ind>` individual constant with a `<Var>`, but there is no means in RuleML to express the condition that the name of the individual constant should be the same character sequence as the lexical value of the data constant.

The first step in preparing the MXSL instance that expresses such a generalization is to apply the utility stylesheet `xsl/ruleml2mxsl.xsl`.

The `<ruleml:Ind>` element from the RuleML above is expressed in MXSL as

```
<xsl:element name="Ind"
  namespace="http://ruleml.org/spec">
  <xsl:copy-of select="'stringParam'"/>
</xsl:element>
```

An MXSL file is self-contained; that is, it is a stylesheet that takes itself as input. As written, the `<xsl:element>` declaration above will execute exactly once. In order to insert multiple `<ruleml:Ind>` elements into the output tree with different names, the content of the `<ruleml:Ind>` element may be generalized in MXSL as

```
<xsl:for-each select="$input/v:first/m:binding">
  <xsl:variable name="stringParam"
    select="v:stringParam" xsi:type="xs:string"/>
  ...
  <xsl:element name="Ind"
    namespace="http://ruleml.org/spec">
    <xsl:copy-of select="$stringParam/node()"/>
  </xsl:element>
  ...
</xsl:for-each>
```

An `<xsl:for-each>` contains an MXSL fragment that will be specialized multiple times, with different values substituted for the iteration parameter (`$stringParam` in the above case). The multiple specializations will be added to the output tree in the order in which the bindings appear in the input variable, as described in the next paragraph. In the most general form of MXSL, the names and values of elements, attributes, processing instructions, and comments, may be generalized in this fashion.

Each `<xsl:for-each>` element should have a different designation, such as `v:first` above so that the variable bindings for the iteration may be uniquely identified. The XPath expression `$input/v:first/m:binding` is then used when "self-executing" an MXSL document to identify the parameter values to be substituted. A binding specification of the example above, with two different bindings for the parameter `stringParam`, takes the form

```
<variable name="input">
  <v:first>
    <m:binding>
      <v:stringParam xsi:type="xs:string"
        >a</v:stringParam>
    </m:binding>
    <m:binding>
      <v:stringParam xsi:type="xs:string"
        >b</v:stringParam>
    </m:binding>
  </v:first>
</variable>
```

Each MXSL file contains a self-referential `xsl-processing` instruction to initiate the XSLT transformation.

```
<?xml-stylesheet type="text/xsl" href="kb2.mxsl"?>
```

After XSLT processing of the MXSL example above, the output contains two formulas (with default RuleML namespace)

```
<Equal>
  <Expr>
    <Fun iri="http://athant.com/mxsl/vocab#constant"/>
    <Data xsi:type="xs:string">a</Data>
  </Expr>
  <Ind>a</Ind>
</Equal>
<Equal>
  <Expr>
    <Fun iri="http://athant.com/mxsl/vocab#constant"/>
    <Data xsi:type="xs:string">b</Data>
  </Expr>
  <Ind>b</Ind>
</Equal>
```

Notice that XSLT processing applied to the `<xsl:for-each>` element leads to multiple `<Equal>` elements occurring as siblings at the same depth in the tree rather than as a nesting to deeper levels of the tree. Recursive nesting, while possible in full XSLT through nesting of `<xsl:call-template>`, is not implemented in MXSL. The complete files for this example are available in the directory `examples/kb2`.

4 Example: Structure Data

The next example illustrates the specification of semantics for structured data. Consider an XML database

```
<ex:records
  xml:base="http://example.com/base">
  <ex:record id="14507" type="business">
    <ex:name>Tara Athan</ex:name>
    <ex:name>Athan Services</ex:name>
  </ex:record>
  <ex:record id="25478" type="customer">
    <ex:name>Susan Smith</ex:name>
    <ex:name>Susie</ex:name>
  </ex:record>
</ex:records>
```

The fields have a context sensitive semantics. The first child `<ex:name>` element indicates of full name, while the second child `<ex:name>` represents either the business name or a nickname, depending on the value of the attribute `type`. To associate such an XML instance with RuleML axioms, we use generalizations of the following axiom instances:

```
<Equivalent>
  <Atom>
    <Rel iri="&v;#data-record-business?rel"/>
    <Data>
      <ex:record id="14507"
        xml:base="http://example.com/base">
        <ex:name>Tara Athan</ex:name>
        <ex:name>Athan Services</ex:name>
      </ex:record>
    </Data>
  </Atom>
  <Atom xml:base="http://example.com/base">
    <oid><Ind iri="#b14507?oid"/></oid>
    <Rel iri="&v;#atom-record-business?rel"/>
    <slot>
      <Ind iri="#name?key"/>
      <Data xsi:type="xs:string">Tara Athan</Data>
    </slot>
    <slot>
      <Ind iri="#business-name?key"/>
      <Data xsi:type="xs:string">Athan Services</Data>
    </slot>
  </Atom>
</Equivalent>
```


The strings 14507, <http://example.com/base>, Tara Athan, Athan Services are generalized by parameters with simple types, as in the previous example. The only new feature is the use of the datatype `xs:anyURI` for the IRI. A similar axiom schema is used for the customer record. The third axiom schema handles the records collection:

```
<Equivalent>
  <Atom>
    <Rel iri="&v;#data-records?rel"/>
    <Data>
      <ex:records>
        <ex:record type="business">
          <ex:record-content/>
        </ex:record>
      </ex:records>
    </Data>
  </Atom>
  <And>
    <Atom>
      <Rel iri="&v;#atom-record-business?rel"/>
      <Data>
        <ex:record type="business">
          <ex:record-content/>
        </ex:record>
      </Data>
    </Atom>
    <Atom>
      <Rel iri="&v;#data-records?rel"/>
      <Data>
        <ex:records>
          <ex:repo/>
        </ex:records>
      </Data>
    </Atom>
  </And>
</Equivalent>
```

This case requires the parameterization of `<ex:record-content/>` and `<ex:repo/>`, which have complex datatypes, where the parameter has datatype `xs:anyType`. The variable bindings are only slightly modified

```
<ex:records>
  <m:binding>
    <ex:record-content xsi:type="xs:anyType">
      <ex:a/>
    </ex:record-content>
    <ex:repo xsi:type="xs:anyType"><ex:b/></ex:repo>
  </m:binding>
</ex:records>
```

Note that the datatype `xs:anyType` can be replaced with user-specified complex types, allowing validation of bindings against a schema as a pre-processing step. All files for this example are available in the directory `examples/kb5`.

The value that is added by the axiom schema is the ability to query the original XML database through the semantic representation in RuleML. Target applications include webservers such as Geoserver [14], which has the capability to serve geographic data that is transformed according to “app-schemas” from the content of one or more databases or other webservice. However, the ability to query the transformed output is lost because Geoserver is unable to invert the app-schema transformation. Because geographic databases tend to be quite larger, the user is burdened with the download of the large transformed database even when they are interested in a small amount of information. The MXSL metalanguage provides a means to express these transformations in a manner that a reasoner can use to invert the transformation, obtaining the queries to be applied to the original datasets. Only certain kinds of transformations are amenable to solution by the reasoning algorithms currently available, and the general case will require the integration of reasoning with numerical solvers. However, many such transformations involve simple re-structuring of data and would be readily handled with such an approach.

5 Semantics for the Finitary Case

The semantics for the finitary case is based on the operational semantics of the XSLT processor, applied to the case when a non-empty but finite set of bindings is specified for all parameters. In this case, the output from XSLT processing of the MXSL file is guaranteed to produce well-formed XML, but not necessarily valid RuleML. Therefore, something more is required to fully define the semantics.

One approach for addressing this issue would be to require the user or generating program to pre-process the bindings to exclude those that lead to invalid RuleML. However, this would require complex conditionals to be generated for the input parameters, if it is even possible to specify such conditions. Instead, we define the meta-interpretation transformation so that the invalid fragments are ignored (as described in the next paragraph), acting as an implicit syntactic-validation-based post-processor.

The implicit post-processing step will be described for the special case, called the “Rulebase-for-each” restriction, when every `<xsl:for-each>` element is the child of the MXSL equivalent of a `<ruleml:Rulebase>` element; that is

```
<xsl:element name="Rulebase" namespace="http://ruleml.org/spec">
```

We consider each unit of XML produced by executing one iteration of XSLT processing on a single `<xsl:for-each>` element as a pre-formula. Any pre-formula that does not satisfy the (finitary) validity requirements for the syntax category of formulas is discarded, while all others are added to the XSLT output in the usual way (as siblings.)

It is an important consequence of the MXSL syntax that such invalid pre-formulas may be identified from the output of a standard XSLT processor applied to an MXSL

instance. This is due to the removal of the XSLT feature (`<xsl:text disable-output-escaping="yes">`) that allows unescaped XML punctuation characters (`<>'"&`) to be inserted implicitly through parameters, rather than explicitly with `<xsl:element>`. Because of the removal of this feature, every pre-formula is a well-formed XML child of a `<ruleml:Rulebase>` element. Therefore the post-processing step may be implemented on the output of a standard XSLT processor by checking the validity of each child of every `<ruleml:Rulebase>` element against the RuleML schema (for the appropriate RuleML sublanguage.)²

6 Semantics for the Infinitary Case

In the case when the input set of variable bindings for a particular `<xsl:for-each>` element is empty, we extend the operational semantics of the metalanguage to interpret the `<xsl:for-each>` element as an axiom schema. The theory of infinitary logic may be used to make this precise.

Again, we restrict our consideration to the "Rulebase-for-each"-restricted case. The default semantics of `<ruleml:Rulebase>` is an implicit conjunction, so that the theory of infinitary logics is applicable [15], whereby it is known that if the base (finitary) logic is first-order then the corresponding infinitary language with countable conjunctions and disjunctions but only finite sequences of quantifiers is complete with respect to the extended deductive apparatus that allows countably infinite sets of axioms and countably infinite proofs³, but is not compact.

In more practical terms, the usual application is when a finitary knowledge base is extended with a finite number of axiom schemas. It is sufficient for reasoning purposes to consider only those instances of the axiom schemas that contain names from the vocabulary of the original finitary knowledge base. Consider, for example, the axioms schema for captured strings presented in Section 3. Forward reasoning of a finitary RuleML knowledge base yields at most a finite number of occurrences of names of the form

```
<ruleml:Ind>...</ruleml:Ind>
```

or expressions of the form

```
<ruleml:Expr>
  <ruleml:Fun
    iri="http://athant.com/mxsl/vocab#constant"/>
    ...
  </ruleml:Expr>
```

² <http://ruleml.org/spec>

³ According to Moore, G.H.: *The Emergence of First-Order Logic*. In P.K. William Aspray (ed.) *History and Philosophy of Modern Mathematics, Volume 11*. University of Minnesota Press, Minneapolis (1988), this result was first proved by Skolem in 1920 Skolem, T.: *Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über dichte Mengen*. *Videnskapsselskapet Skrifter, I. Matematisk-naturvidenskabelig Klasse 6*. p. 1–36 (1920).

Therefore it is sufficient to generate a finite number of equations, corresponding to specializations of the axiom schema for the particular, finite, set of bindings relevant to the knowledge base. Because this axiom schema does not create any new names or expressions, there is no need to revisit the axiom schema during forward reasoning on the knowledge base extended by this finite set of equations. Therefore, this axiom schema does not undermine the finite-model property of the knowledge base to which it is added.

In general, with proper design of the axiom schemas and naming conventions, i.e. avoiding an infinite cascade of new names or expressions, the set of relevant specializations of the axiom schemas will be finite, ensuring that either backward or forward reasoning on such extended knowledge bases will terminate provided the base logic has this property.

7 Extensions

The "Rulebase-for-each" restriction of MXSL may be safely relaxed to allow a `<ruleml:Assert> <xsl:for-each> <ruleml:Query>` tree. This is because such queries can be transformed to an equivalent "Rulebase-for-each" case by forming the negation of the existential closure of the formula contained in each query and inserting this into the rulebase which is being queried, where success of the query is equated with inconsistency of the resulting rulebase.

Further relaxation of the metalanguage incurs the risk of allowing the expression of infinite sequences of quantifiers. For example, suppose we allow a `<ruleml:And> <xsl:for-each>` tree. It is then possible to construct an instance such that every `<xsl:for-each>` iteration produces a new free variable, named according to value of the iteration parameter. For example,

```
<Exists closure="universal">
  <Var>F</Var>
  <And>
    <Atom>
      <Rel>Apply</Rel>
      <Var>F</Var>
      <Var>M1</Var>
      <Data>1</Data>
    </Atom>
  </And>
</Exists>
```

If the "And-for-each" construction is allowed, then we may create from the RuleML axiom above an MXSL axiom schema by generalizing the natural number 1 that appears both in the name of the variable `<Var>M1</Var>` and the content of the data element `<Data>1</Data>`. The resulting axiom schema can be applied to all natural numbers through the use of the datatype `xs:positiveInteger` for the schema parameter. See `examples/andForeach/andForeach/mxsl` for the complete

axiom schema. The effect of the `closure` attribute is an infinite sequence of quantifiers, as this is necessary to implement closure on an infinite number of free variables. With this axiom schema we are well on our way to a characterization of the real numbers, and have thus left the realm of first-order logic and countable models, let alone finite models. Thus we cannot consider the “And-for-each” relaxation as a safe extension of MXSL.

Another possible extension of MXSL would be to allow recursive application of the metatransformation until a fixed point is reached, as was considered in [8]. Such recursion would implement a meta-circular evaluator [18]. In general it is an open question as to how such extensions of the metalanguage would affect the logical properties of the resulting infinitary language.

8 Metalanguage Abstract Syntax

From the development and theory explored while developing MXSL, and described above, we have derived the following abstract syntax principles to form the foundation of a metalanguage development approach that is applicable to other KR formats, whether XML-based or not.

- The metalanguage uses a syntactic model of the base language, so that, for example, grouping symbols are entered as pairs through a single command.
- The hierarchical structure of a metalanguage instance mirrors the structure of the output document.
- An iteration command is available to generate a set of sibling items based on a pattern, instantiated with parameters.
- Text is always added through a metalanguage command, which implements output-escaping of punctuation characters appropriate to the base language.

All of the above can be implemented within XSLT for an arbitrary format. For example, a stylesheet with special purpose templates for an XSLT-based metalanguage for CLIF [19] has been implemented in `examples.cl1/cl-header.xml`. This stylesheet may be imported into an MXSL-CLIF instance, allowing templates to be called, implementing, for example, the MXSL-CLIF extension syntax equivalent to `<xsl:element>`. That is, instead of XML tags, the expression is wrapped with parentheses and the name of the CLIF 'element' is placed into the first position of the list. MXSL-CLIF also has templates for the single quote and double quote containers for character sequences.

Of special importance is the handling of escaped punctuation. As was described in the previous section, proper handling of grouping and tokenizing symbols is a key feature for the output of the XSLT transformation to be parsable into pre-formulas, which will then be individually post-processed based on their syntactic validity in the base language. The imported MXSL-CLIF stylesheet uses the XSLT 2.0 character map and the XPath `translate` function to handling the escaping of the characters ' " () \ that may appear in the value of iteration parameters.

An example illustrating the CLIF axiom schema for the captured string syntax is given in `examples/cl1/cl1.mxsl`. As an excerpt, the syntax for the MXSL command to wrap a character sequence in double quotes looks like

```
<xsl:call-template name="element-dquote">
  <xsl:with-param name="arg"
    select="$stringParam/node()" />
</xsl:call-template>
```

Similar extensions to MXSL are possible for other formats, such as JSON. Although it appears awkward to have an XML metalanguage for rather different formats, such as CLIF or JSON, there are several advantages to this approach. For example, the common XML format of the metalanguages permits CLIF texts to be embedded into the RuleML XML syntax as data, while the semantic content is retained by defining axiom schemas relating parameterized CLIF sentences to the associated RuleML formulas.

Further, XML has an advantage that many of these other formats lack: prefixes and namespaces. The prefixes of the XML element names, and their associated namespaces, allow us to distinguish the components of the metalanguage from the components of the base language and avoid naming collisions.

9 Conclusions

We have shown that a (small) subset of the XSLT language, which we call MXSL can be used as a flexible and powerful metalanguage for XML-based KR languages, with the additional advantage of familiarity to many XML users. This metalanguage may be used to express axiom schemas, the semantics of structured data and generalized queries. The paradigm used to develop the metalanguage for the XML syntax can be applied to non-XML-based formats. Future investigations will explore the implementation of reasoning using the MXSL metalanguage.

References

1. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures W3C Working Draft (December 3, 2009), <http://www.w3.org/TR/xmlschema11-1/>
2. ISO/IEC: 19757-2 Document Schema Definition Language (DSDL) Part 2: Regular-grammar-based validation - RELAX NG. International Organization for Standardization, Geneva (2008)
3. ISO: 19757-3 Information technology – Document Schema Definition Language (DSDL) – Part 3: Rule-based validation – Schematron. International Organization for Standardization, Geneva (2006)
4. Semantic Web on XML: Architecture, <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>
5. Introducing JSON, <http://json.org/>

6. A Web Data Model Unifying XML and RDF,
<http://www.dfki.uni-kl.de/~boley/xmlrdf.html>
7. Patel-Schneider, P., et al.: The Yin/Yang web: XML syntax and RDF semantics. In: Proceedings of the 11th International Conference on World Wide Web, pp. 443–453. ACM, Honolulu (2002)
8. Athan, M.: XCLX: An XML-based Common Logic eXtension with Embedded Geography Markup Language. *Geographic Information Systems*. p. 77. University of Leeds (2011)
9. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: The Overarching Specification of Web Rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S., et al. (eds.) *RuleML 2010*. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010)
10. XSL Transformations (XSLT) Version 1.0, <http://www.w3.org/TR/xslt>
11. XQuery 1.0 and XPath 2.0 Data Model (XDM), 2nd edn.,
<http://www.w3.org/TR/xpath-datamodel/>
12. Universal Turing Machine in XSLT, <http://www.unidex.com/turing/utm.htm>
13. IKL Guide, <http://www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html>
14. Welcome - Geoserver, <http://geoserver.org/display/GEOS/Welcome>
15. Infinitary Logic,
<http://plato.stanford.edu/entries/logic-infinitary/>
16. Moore, G.H.: The Emergence of First-Order Logic. In: William Aspray, P.K. (ed.) *History and Philosophy of Modern Mathematics*, vol. 11. University of Minnesota Press, Minneapolis (1988)
17. Skolem, T.: Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über dichte Mengen. *Videnskapsselskapet Skrifter, I. Matematisk-naturvidenskabelig Klasse 6*, 1–36 (1920)
18. The Metacircular Evaluator,
http://mitpress.mit.edu/sicp/full-text/book/book-Z-H-26.html#%_sec_4.1
19. ISO/IEC: Information technology — Common Logic (CL): a framework for a family of logic based languages. International Organization for Standardization, Geneva (2007)

An Approach to Parallel Class Expression Learning

An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland

School of Engineering and Advanced Technology, Massey University
Palmerston North, New Zealand
{a.c.tran,j.b.dietrich,h.w.guesgen,s.r.marsland}@massey.ac.nz

Abstract. We propose a Parallel Class Expression Learning algorithm that is inspired by the OWL Class Expression Learner (OCEL) and its extension – Class Expression Learning for Ontology Engineering (CELOE) – proposed by Lehmann et al. in the DL-Learner framework. Our algorithm separates the computation of *partial definitions* from the aggregation of those solutions to an overall *complete definition*, which lends itself to parallelisation. Our algorithm is implemented based on the DL-Learner infrastructure and evaluated using a selection of datasets that have been used in other ILP systems. It is shown that the proposed algorithm is suitable for learning problems that can only be solved by complex (long) definitions. Our approach is part of an ontology-based abnormality detection framework that is developed to be used in smart homes.

Keywords: description logic learning, class expression learning, parcel, parallel learning, abnormal behaviour detection.

1 Introduction

Description logic (DL) is a popular formalism used in knowledge representation. Amongst its strengths are the availability of a formal semantics, the standardisation of description-logic-based languages by the W3C (RDFS and several versions and flavours of OWL [1]), and the availability of robust tools to edit and reason about ontologies.

The primary problem that motivates our research is the classification of normal and abnormal activities in a smart home environment, where elderly people are monitored by a system that can alert medical professionals if abnormal behaviour is detected [2]. It is important in this scenario that we do not miss any abnormal behaviours, in particular if these behaviours potentially pose a threat to the person living in the smart home. In technical terms, this means that we aim at avoiding false positives in the class of normal behaviours.

Using a symbolic (logic-based) approach in this context has the advantage that systems can be designed that are inherently more trustworthy than sub-symbolic machine learning approaches, as system decisions are traceable through the proofs associated with classifications.

A common problem in symbolic AI is to find the “right” set of rules. Here, by rules we mean the expressions that define concepts such as normal and abnormal

behaviour. It takes a significant effort to create and maintain such a set of rules, and comprehensive validation against real world data is needed to assess its accuracy. An alternative approach is to learn the rules directly from sample datasets. This has several advantages: if it can be demonstrated by means of a formal proof that the learning algorithm produces expected rules and all available data have been fed into the learning algorithm, then validation is no longer necessary. Also, if new training data becomes available, the algorithm can be easily reapplied and the new definitions can be created. In other words, the system can easily be re-calibrated as needed.

However, this implies that we need to apply the learning algorithm often, and on large datasets. Therefore, the scalability of the algorithm becomes a major concern. Benchmark tests performed by Hellmann [3] indicate that the DL-Learner [4] is a suitable starting point for the development of an expressive and scalable DL learning algorithm. Our experiments with the use cases described in [2] show that CELOE and its ancestor – OCEL – are the most suitable algorithms for solving this problem amongst the algorithms implemented in DL-Learner. These algorithms generate expressions of increasing complexity starting with explicitly defined classes in the ontology, and assess the accuracy of these expressions against a training set consisting of positive and negative examples. If an expression with a sufficiently high accuracy score is found, the algorithm terminates and the expression is returned as the result of the computation.

Unfortunately, our experiments suggest that these algorithms do not have the level of scalability necessary to be used in the smart home application domain, particularly for smart home datasets generated from the uses cases in [2]. An analysis of these algorithms reveals that the accuracy measure used to direct the generation and evaluation of descriptions is a combination of correctness (no negative examples are covered by the computed expression) and completeness (all positive examples are covered). Motivated by the need for a higher accuracy in learning normal behaviour, we propose a DL learning algorithm that separates this process into two steps: first the generation of correct rules (i.e. they do not cover any negative examples) but not necessarily cover all positive examples, and then the aggregation of those rules into a (sufficiently) complete solution. In addition, there is no need to serialise these two steps: they can be performed concurrently. In particular, multiple branches within the tree of possible descriptions can be traversed concurrently by multiple workers to find partial results, while a central reducer aggregates partial results to the overall solution until all positive examples are covered. The reducer also has the responsibility of removing redundant definitions covering overlapping sets of positive examples. We discuss several strategies to do this in section 3.

This approach follows the general ideas of the map-reduce architecture [5] and therefore lends itself to parallelisation using either multiple threads that can take advantage of multi-core processors, or may be developed for cloud computing platforms such as Amazon EC2 in the future. It also has the advantage that the resulting system shows anytime characteristics [6], which means that: i) it

can return a correct solution even if it is interrupted before a complete solution is computed, and ii) the solution is expected to improve (i.e., become more complete) with increasing runtime of the system.

2 Related Work

Description logic learning has its roots in inductive logic programming (ILP) [7,8,9]. In ILP, sets of positive and negative facts and some background knowledge are given, and an ILP algorithm is used to compute a logic program that describes all the positive and none of the negative examples. There are two fundamentally different strategies to compute this program: top-down and bottom-up [7]. Combined strategies have also been investigated by different authors [10].

In description logic learning we are interested to find concepts that describe all given positive examples, but do not describe any of the negative examples. Our work is directly based on the DL-Learner framework [4], particularly the CELOE and OCEL algorithms. These algorithms use a top-down strategy to learn concepts. Starting with the root of the concept class hierarchy, concepts are refined by means of specialisation until a suitable concept is found. The description learning space expansion is mainly directed by the accuracy (a combination of correctness and completeness) of concepts with respect to the positive and negative examples, the complexity of the expressions, the accuracy gained in each expansion step, and some other factors. Note that the refinement operator used by these algorithms also has some implicit support for the bottom-up strategy as it will generate complex expressions using disjunctions.

Lisi [11] has proposed an alternative top-down approach based on the hybrid \mathcal{AL} -log language which combines \mathcal{ALC} description logic and Datalog for knowledge representation. This makes it possible to learn Datalog rules on top of ontologies.

Several other approaches to concept learning have been proposed. This includes LCSLearn [12], an early bottom-up approach that creates concepts by joining most specific concepts created for individuals (positive examples) using disjunction. This is a very simplistic approach that creates large concept definitions that are not truly intentional in a sense that those definitions are only enumerations of the sets of individuals they define. YinYang [13] is a hybrid learner that uses a combination of bottom-up (starting from most specific concepts) generalisation and top-down specialisation strategies.

Our contribution is similar to DL-FOIL [14]: we separate the computation of partial correct concepts from the computation of a complete concept. There are two main differences however: (i) the algorithm proposed by [14] is serial by nature as the computation of the partial correct concepts is executed in an inner loop, while we use a parallel computation model, and (ii) we propose an extra reduction step to compute an optimal set of partial concepts to be used in order to compute the overall (complete) result. The above differences aim to bring some

benefits: i) improved scalability of the algorithm due to parallelisation, ii) the any time characteristic of the algorithm, which means that the algorithm can produce correct (but not necessarily complete) solutions if interrupted prematurely, and iii) the flexibility gained through a separate reduction step that allows us to tradeoff completeness, number of partial definitions and the (average) lengths of the partial definitions.

3 Algorithm

Our algorithm is inspired by the popular map-reduce framework [5] that is widely used to process large amounts of data. Here, input data to be processed is divided into several pieces (sub-problems) and processed by multiple workers (map step) in parallel and then the intermediate results are aggregated (reduce step) into a final result.

In the context of our work, the problem of finding a correct and complete concept definition is mapped to workers responsible for refining and evaluating candidate concepts. The actual refinement operator used for this is the refinement operator proposed in [15], which is one of the refinement operators currently supported by the DL-Learner framework. However, the operator is customised by disabling the generation of disjunctions as the generalisation is done in a separate reducer step. In addition, its numerical data properties refinement has also been improved by using a better strategy for identifying the domain of the refinement.

The reduce step consists of combining the partial definitions until a complete (or at least sufficiently good) coverage of the positive examples is obtained. Often, this yields a set of partial definitions that is redundant in the sense that multiple definitions cover the same positive examples, and that a proper subset of definitions exists that is also complete with respect to the positive examples. Here, we propose the use of a set coverage algorithm [16] to find such a subset. This allows us to tailor the main algorithm, e.g. in order to compute smaller sets of concepts, or sets of concepts with a shorter average expression length (lower complexity).

An informal illustration of the algorithm is given in Figure 1. It shows the interaction between the two parts of the algorithm: the reducer that aggregates and compacts the partial definitions, and the worker(s) producing the partial definitions. The coordination is done using an agenda. The agenda contains the concepts to be refined, and an ordering of its nodes generated by an expansion heuristic. This means that there is always a top element representing the most promising concept for refinements based on the heuristic used. This element is assigned to workers for processing. The current search heuristic is based on the heuristic used in [17] that associates concepts with a score mainly based on their accuracy (combination of correctness and completeness). In addition, a level of penalty on complexity of the concepts (short expressions are preferred), bonus on accuracy gained, etc. are also applied. In our learning heuristic, we also penalise long descriptions to avoid infinite deep searches because the refinement

operator used in our learner is infinite. Instead, a description’s score is mainly based on the correctness. Note that our learning heuristic can help to avoid infinite deep searches, but this does not avoid infinite loops in the whole learning process because the refinement operator itself is infinite. Therefore, the termination is controlled by the accuracy of the definitions generated and the timeout mechanism.

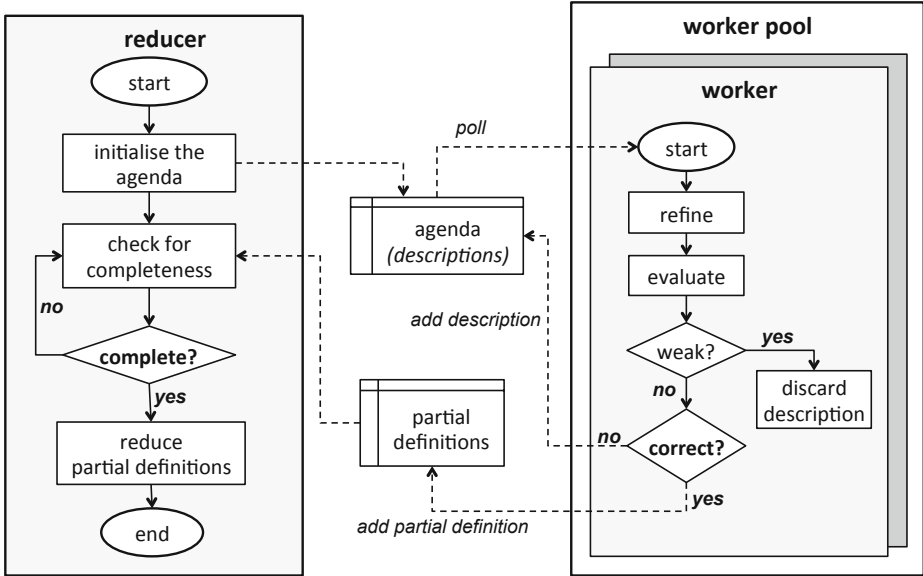


Fig. 1. Reducer-Workers interaction

For a more formal definition, we introduce some notations first. A *learning problem* is a structure (K, E^+, E^-) that consists of a knowledge base K , a set of positive examples E^+ and a set of negative examples E^- . We say that a concept C covers an example e iff $K \models C(e)$. A concept C is called *correct* if it does not cover any negative example and *weak* if it covers none of the positive examples. We also refer to correct concepts as *definitions*. A definition is called a *partial definition* if it covers at least one and less than all positive examples, and a *complete definition* if it covers all positive examples.

There are some useful metrics to measure the amounts of correctness and completeness of a concept C . Let $R(C)$ be the set of individuals covered by C . Then $un(C) = E^- \setminus R(C)$ is the set of negative examples not covered by C , and $cp(C) = E^+ \cap R(C)$ the set of covered positive examples. We can then define *correctness*, *completeness* and *accuracy* using predictive accuracy methodology as follows:

$$correctness(C) = \frac{|un(C)|}{|E^-|}$$

$$completeness(C) = \frac{|cp(C)|}{|E^+|}$$

$$accuracy(C) = \frac{|cp(C)| + |un(C)|}{|E^+ \cup E^-|}$$

Our algorithm can now be defined in two parts. The computational heavy part is done by the *multiple* workers: this is the refinement and the evaluation of concepts. In particular, the evaluation of (complex) concepts (i.e., the check whether a given example is defined by a concept) requires an ontology reasoner. By default, Pellet [18] is used for this purpose.

The reducer creates a worker pool, which manages a number of workers, and assigns new concepts for refinements and evaluations to worker pool until the completeness of the combined partial definitions is sufficient. Then the reducer tries to reduce the number of partial definitions in order to remove redundancies using a *reduction* function. While the reducer computes sets of concepts, these sets can be easily aggregated into a single concept using disjunction.

Algorithm 1 (Reducer Algorithm). For a given learning problem (K, E^+, E^-) , a noise value $\varepsilon \in [0, 1]$ and a pool of workers, compute a set of partial correct solutions $\{C_i\}$ such that $completeness(\sqcup_i(C_i)) \geq 1 - \varepsilon$.

```

1: agenda := {⊤}
2: solutions := {}
3: uncovered_positive_examples := E+
4: create a worker_pool
5: while |uncovered_positive_examples| > |E+| × ε do
6:   wait for new partial definition(s) produced by workers
7: reduce(solutions)
8: return solutions

```

The worker algorithm refines a concept and evaluates the results of the refinement. It will first check whether concepts are weak. If this is the case the concept can be safely removed from the computation as no partial definition can be computed through specialisation. If a concept is a partial definition (i.e., correct and not weak), it is added to the (shared) partial definitions set. If a concept is not weak, but also not correct (i.e., if it covers some positive and some negative examples), it is added back to the agenda and therefore scheduled for further refinement. Note that the concepts that have been refined can be scheduled for further refinement. This is necessary as each refinement step only computes a finite (and usually small) number of new concepts, usually constrained by a complexity constraint. For example, a concept of a given size N could first be refined to compute new concepts of a length $N + 1$, and later it could be revisited to compute more concepts of length $N + 2$, etc. This technique is used in the original DL-Learner and discussed in detail in [17]. When implementing workers, an additional redundancy check takes place to make sure that the same concept computed from different branches in the search tree is not added twice to the agenda.

Algorithm 2 (Worker Algorithm). For a set of positive examples E^+ and a set of negative examples E^- , refine a given concept C using a refinement operator ρ , and evaluate the refinements.

```

1: refinements :=  $\rho(C)$ 
2: for all description  $\in$  refinements do
3:   positive_covered := positive examples covered by refinement
4:   if description is not weak then
5:     if description is correct then
6:       add description to solutions
7:       uncovered_positive_examples := uncovered_positive_examples \ positive_covered
8:     else
9:       add description to agenda

```

For the actual reduction step, we have investigated three simple algorithms:

- GMPC (greedy minimise partial definition count)
- GPL (greedy minimise partial definition length)
- GOLR (greedy online algorithm - first in first out)

As the names suggest, they are all greedy optimisation algorithms that are based on sorting the partial definitions. Once the partial definitions are sorted, a new solution set (called the reduction set) is created and solutions are added to this set in descending order. A definition is added only if it covers at least one positive example not yet covered by any other solution in the reduction set. Details are given in algorithm 3.

We have used different sort criteria, resulting in the different algorithms. In GMPC, we sort partial definitions according to the number of positive examples they cover, preferring definitions that cover more positive examples. If two definitions cover the same number of positive examples, we use the lexicographical order of the respective string representations as a tiebreaker. This is important to make the results repeatable. Otherwise the order that is used when iterating over definitions could depend on internal system hash codes which the application does not control.

In GPL, we sort definitions according to their expression lengths, preferring definitions with a shorter length. If two definitions have the same expression lengths, we again use the lexicographical order.

In GOLR, we use time stamps assigned to definitions when they are added to the solutions, preferring definitions that have been added earlier. While the other two heuristics have to be run in batch mode after a complete set of definitions has been computed, this algorithm can be employed just in time, the reduction can take place whenever a new definition is found and added. This algorithm is therefore very space efficient compared to the other two. On the other hand, how timestamps are assigned in an application may depend on thread scheduling. This again cannot be controlled completely by the application, causing variations in the results between runs.

More formally, we define a generic reduction algorithm that is based on a sort function as follows:

Algorithm 3 *Generic greedy reduction algorithm based on sorting.* For a set of definitions D and a function $\text{cover} : D \rightarrow 2^{E^+}$ that associates partial definitions with the sets of covered positive examples, compute a subset $D' \subseteq D$ such that $\bigcup_{d \in D} \text{cover}(d) = \bigcup_{d \in D'} \text{cover}(d) = E^+$

```

1:  $D' := \emptyset$ 
2:  $\text{positive\_covered} := \emptyset$ 
3: sort  $D$ 
4: while  $D$  is not empty and  $\text{positive\_covered} \subset E^+$  do
5:    $d := \text{poll}(D)$ 
6:   if  $\text{cover}(d) \not\subseteq \text{positive\_covered}$  then
7:      $D' := D' \cup d$ 
8:      $\text{positive\_covered} := \text{positive\_covered} \cup \text{cover}(d)$ 
9: return  $D'$ 

```

4 Validation

4.1 Methodology

We have implemented our algorithm using Java. The package is called the ParCEL (PARallel Class Expression Learning) and a minimal set of this package is available at <https://parcel-mu.googlecode.com/>. The algorithm is also integrated into the DL-Learner repository <http://dl-learner.svn.sourceforge.net>.

In the validation, we were not only interested in measuring the overall computation time of the benchmark learning problems, but also in measuring how quickly accuracy improved during the computation. We consider this to be important in scenarios where an application could intercept the learner once a sufficiently complete solution has been computed. For this purpose, two different sorts of experiments have been performed: i) a 10-fold cross validation to measure the learning time and accuracy, ii) a training run on different levels of parallelism to observe the accuracy improvement on the training set. The former follows the standard cross-validation methodology in statistics. In the later experiment, we start a background watcher thread that frequently takes probes from learner thread(s) and records them. This thread represents some overhead, so the net computation times are in fact slightly less than the values given below. We benchmarked our learner against the CELOE and OCEL algorithms.

In our experiments, we have used a number of datasets that have been used by other authors in similar experiments [19,20] to benefit comparisons. All datasets used in this paper, except the UCA1 which will be described later, can be found in any DL-Learner release or in the DL-Learner repository. An overview of these datasets is given in Tables 1 and 2. Note that DL-Learner is in the development process. New revisions have been being issued very regularly and the learning time and accuracy for the same dataset may change over the revisions. In our

Table 1. Experiment datasets summary

	Moral simple	Forte	Poker	Carcino genesis	Family benchmark	UCA1
Classes	43	3	2	142	4	65
Classes assertions	4646	86	374	22,372	606	300
Object properties	0	3	6	4	4	4
Object property assertions	0	251	1080	40,666	728	200
Data properties	0	0	0	15	0	11
Data property assertions	0	0	0	11,185	0	200
Examples	102p/ 100n	23p/ 163n	4p/ 151n	182p/ 155n	-	73p/ 77n

Table 2. Family benchmark datasets - Number of examples

	Aunt	Uncle	Cousin	Daughter	Father	Grandson	Brother
Examples	41p/ 41n	38p/ 38n	71p/ 71n	52p/ 52n	60p/ 60n	30p/ 30n	43p/ 30n

experiment, we have used DL-Learner version 1.0.1. In addition, we used the default learning configuration for CELOE/OCEL and ParCEL for all datasets.

In our experiment, we also used an additional dataset – UCA1 – which is extracted from the use case descriptions of the smart home domain we are primarily interested in [2]. This use case describes an over-long shower scenario in a smart home, in which the showering duration depends upon the season of year. This dataset is supported by an underlying smart home ontology which contains the basic concepts for describing the activities in smart homes and some context information, particularly the temporal and spacial information. The scenario was modelled using a Bayesian Network and then the network was used to generate the simulation dataset. The actual dataset contains a set of showering activities, their start times and durations.

For the experiments, we used a Linux server with a 8 x Intel Xeon E5440 @2.83GHz processor, 32GB memory and the Redhat 4.1.2 (Linux version 2.6.18) operating system with a JRE 1.6.0 (64-bit) Java Virtual Machine (JVM). The heap size of the JMV in our experiments is 5GB.

The length of definition reported is the length of the best description learnt so far.

4.2 Result Summary

Table 3 shows a summary of the results. The reduction mechanism used here is GMPC, i.e. we use a simple greedy algorithm to reduce the number of partial definitions.

Most importantly for our application scenario, ParCEL outperformed CELOE on the UCA1 dataset, both on learning time and accuracy. Although the length of the definition produced by ParCEL is longer than CELOE, it is readability and describes well our scenario. For examples, one of the learnt results for the concept *normal showering* is the disjunction of the following partial definitions:

1. EXISTS activityHasDuration.(hasDurationValue >= 4.5 AND hasDurationValue <= 15.5)
2. EXISTS activityHasDuration.(hasDurationValue >= 15.5 AND hasDurationValue <= 19.5) AND EXISTS activityHasStarttime.Spring
3. EXISTS activityHasDuration.(hasDurationValue >= 15.5 AND <= 19.5) AND EXISTS activityHasStarttime.Summer
4. EXISTS activityHasStarttime.Autumn AND ALL activityHasDuration.(hasDurationValue >= 4.5 AND hasDurationValue <= 19.5)

One of the most difficult learning problems in our experiment is the Carcino-Genesis dataset. Learning results for this dataset reported in [14,20] show that CELOE gives the best accuracy in comparison with other learners with a certain learning configuration. In our experiment, neither CELOE nor ParCEL could find an accurate definition on the training dataset before they ran out of memory. CELOE runs out of memory in around 2100 seconds and ParCEL can run for approximately 15800 seconds with the same JVM heap size. The experiment result shows that ParCEL outperformed CELOE on the training dataset by 36%. However, the accuracy for the testing dataset is not significantly different: $54.618\% \pm 2.711\%$ for CELOE and $55.597\% \pm 9.516\%$ for ParCEL. The above accuracy is obtained at 2000 seconds when CELOE is approaching the out of memory exception. Although ParCEL can run for more than 15800 seconds, we only let it run the same amount of time as CELOE since our experiments demonstrate that the accuracy does not improve significantly for the longer runs. Note that this result is generated by the default learning configuration and it may be different for the refined learning configuration. For example, show that the predictive accuracy can be improved by allowing a level of noise in training dataset. However, this has not yet been studied in our research.

A paired t-test rejected the null hypothesis (that the running times came from the same distribution) at the 5% confidence level, for both the running times and accuracies in Table 3. However, while an F test showed that the accuracies were normally distributed, this was not true for the running times, and so this result should be treated with caution.

4.3 Performance Improvement Comparison

We have used a monitoring thread as described above to investigate the level of approximation that the learners can achieve. This is shown in figure 2. The slightly odd values on the x -axis are due to the fact that they were taken from the timestamps when the monitoring thread returns data. CELOE computes a solution of about 0.55 accuracy very quickly (the first probe already returns this values), but then “stays flat”. On the other hand, the ParCEL almost reaches

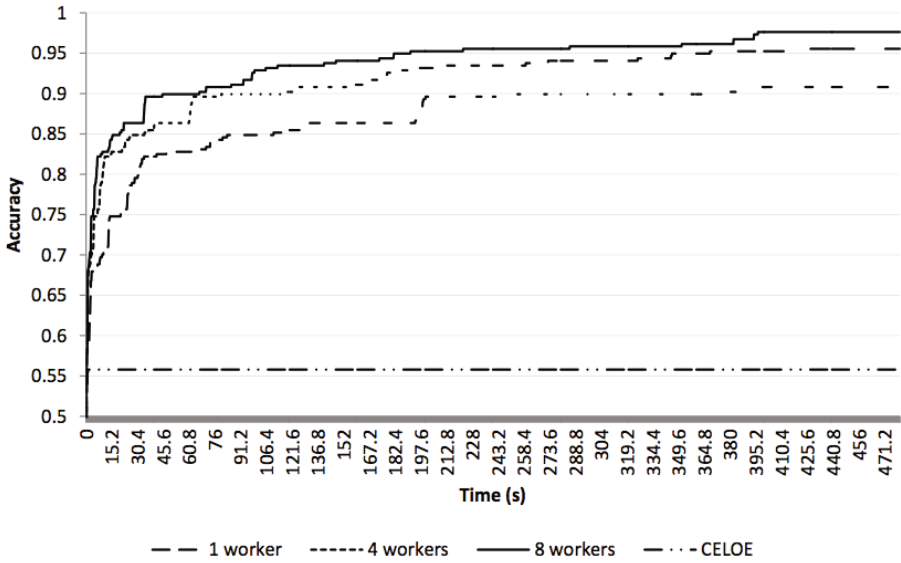


Fig. 2. Learning using the Carcino Genesis dataset

maximum accuracy, i.e., a level of completeness of more than 0.95. The figure also shows the impact of the number of threads: adding more threads can speed up the computation.

Figure 3 shows details for the UCA1 dataset. In this case, CELOE cannot compute a very accurate result before it times out, whereas ParCEL succeeds. Adding more threads can again speed up the computation significantly.



Fig. 3. Learning using the UCA1 dataset

4.4 Definition Aggregation

Finally, we conducted an experiment to compare the three reduction algorithms discussed earlier. Here we measured the length of descriptions and the number of descriptions defined. To compare the description length, we use the method defined above, i.e., we measure the lengths of the virtual disjunction we could create from the set of partial definitions. Comparison between reduction strategies is given in Table 4. The results show that GMPC gives the shortest definition in most of the experiments and thus the definitions are more readable. On the other hand, GORL produces the longest definitions in all the experiments. However, GMPC requires all partial definitions to be kept until the learning finishes while the GORL can perform the reduction on the fly when the learning is happening. This may give us a selection on the tradeoff between the readability of the learnt definition and the memory used by the learner as well as the learning time.

Table 4. Definition length comparison between algorithms and reduction strategies (averages \pm standard deviations of 10 folds)

dataset	CELOE	GMPC		GORL		GMPL	
	def. length	no. of partial def.	avg. partial def. length	no. of partial def.	avg. partial def. length	no. of partial def.	avg. partial def. length
Moral	3 \pm 0	2.1 \pm 0.316	1.517 \pm 0.053	3.4 \pm 0.966	2.15 \pm 1.263	3 \pm 0	1.667 \pm 0
Forte	12 \pm 1.054	1.9 \pm 0.738	9.167 \pm 3.15	2.3 \pm 0.675	7.708 \pm 0.429	2.3 \pm 0.675	7.617 \pm 0.209
Poker-straight	11.7 \pm 0.675	1.7 \pm 0.675	10.9 \pm 1.308	2.7 \pm 0.483	9.883 \pm 1.457	2.7 \pm 0.483	9.483 \pm 0.976
UCA1	9 \pm 0 @OOMem	4 \pm 0	12.75 \pm 0	9.7 \pm 1.059	13.479 \pm 0.217	5.5 \pm 1.179	13.063 \pm 0.466
Aunt	19 \pm 0	3.1 \pm 0.316	8.267 \pm 0.492	8.9 \pm 2.079	7.75 \pm 0.289	8.3 \pm 1.418	7.477 \pm 0.308
Brother	6 \pm 0	1 \pm 0	5.2 \pm 0.422	1 \pm 0	5.6 \pm 0.516	1 \pm 0	5.1 \pm 0.316
Uncle	19 \pm 14.94	3 \pm 0	8.4 \pm 0.378	7.1 \pm 1.287	7.917 \pm 0.163	6.8 \pm 1.135	7.746 \pm 0.482
Cousin	23.4 \pm 2.591	2 \pm 0	8.5 \pm 0	8.2 \pm 4.158	8.5 \pm 0.575	5.7 \pm 1.252	8.095 \pm 0.208
Daughter	5 \pm 0	1.1 \pm 0.316	5.25 \pm 1.087	1.5 \pm 0.527	7.55 \pm 2.409	1.4 \pm 0.843	5.333 \pm 0.471
Father	5 \pm 0	1 \pm 0	5.5 \pm 0.527	1 \pm 0	5.2 \pm 0.422	1 \pm 0	5.3 \pm 0.483
Grandson	7 \pm 0	1.3 \pm 0.483	7.4 \pm 0.459	2.9 \pm 0.568	7.525 \pm 0.553	2.5 \pm 0.707	7.2 \pm 0.502

5 Conclusion

Our approach to parallelising the class expression logic learning shows promising results on the datasets used in the evaluation. By dividing the learning process into two separate stages, one for generating correct but potentially incomplete definitions and another one for aggregating the partial definition to a complete (or nearly complete) solution, we were able to spread the task over several sub-processes that can run in parallel. As a result, we are able to utilise multi-core machines and potentially also cloud computing, which makes the task of description logic learning more scalable.

Since the aggregation of partial solutions is now not integrated in the refinement procedure anymore but runs as a separate thread concurrently to it, we are able to easily test different strategies for aggregating the partial definitions. The ones that we have tested are greedy strategies which avoid exhaustive search for an optimal aggregate and therefore scale more easily.

The main motivation for our research is the classification of normal and abnormal activities in a smart home environment, in which UCA1 is one of our simulation datasets. With this dataset, DL-Learner gave the best solution with 91.2% accuracy before it ran out of memory (with 5GB heap space allocated and 38 minutes run time). Describing this problem requires a description with the minimal length around 42 to 73 and this may be one of the potential causes that exploded DL-Learner memory. Generally, 91.2% accuracy is a good learning result. However, in this application domain, any false positive or false negative classifications may affect strongly on the inhabitant safety and thus an accurate definition is preferred to a readable one. With the combination of specialisation and generalisation and the parallelisation approach, we are preliminarily getting success with the first datasets in our research. In addition, completeness of the partial definitions may provide us an interesting dimension in our classification: belief of the classification.

In most of the datasets in our experiment, our learner algorithm gives a promising result both in accuracy and learning time. The only dataset that our learner could not give a better result is CarcinoGenesis. It shows that our learning currently does not deal well with noise data and this is a future development for our learner so that it can deal with various learning problems.

References

1. McGuinness, D., Van Harmelen, F., et al.: OWL web ontology language overview. W3C Recommendation 10, (2004) 2004-03
2. Tran, A.C., Marsland, S., Dietrich, J., Guesgen, H.W., Lyons, P.: Use Cases for Abnormal Behaviour Detection in Smart Homes. In: Lee, Y., Bien, Z.Z., Mokhtari, M., Kim, J.T., Park, M., Kim, J., Lee, H., Khalil, I. (eds.) ICOST 2010. LNCS, vol. 6159, pp. 144–151. Springer, Heidelberg (2010)
3. Hellmann, S., Lehmann, J., Auer, S.: Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst.* 5(2), 25–48 (2009)
4. Lehmann, J.: DL-Learner: learning concepts in description logics. *The Journal of Machine Learning Research* 10, 2639–2642 (2009)

5. Ghemawat, S., Dean, J.: Mapreduce: Simplified data processing on large clusters. In: Symposium on Operating System Design and Implementation, OSDI 2004, San Francisco, CA, USA (2004)
6. Zilberstein, S.: Using anytime algorithms in intelligent systems. *AI Magazine* 17(3), 73–83 (1996)
7. Muggleton, S.: Inductive logic programming. *New Generation Computing* 8(4), 295–318 (1991)
8. d’Amato, C., Fanizzi, N., Esposito, F.: Inductive learning for the semantic web: What does it buy? *Semantic Web* 1(1), 53–59 (2010)
9. Lavrac, N., Dzeroski, S.: *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York (1994)
10. Zelle, J.M., Mooney, R.J., Konvisser, J.B.: Combining top-down and bottom-up techniques in inductive logic programming. In: Proceedings of the 11th International Conference on Machine Learning, pp. 343–351 (1994)
11. Lisi, F.A.: Building rules on top of ontologies for the semantic web with inductive logic programming. *Theory and Practice of Logic Programming* 8(3), 271–300 (2008)
12. Cohen, W., Hirsh, H.: Learning the classic description logic: Theoretical and experimental results. In: Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Citeseer, pp. 121–133 (1994)
13. Iannone, L., Palmisano, I.: An Algorithm Based on Counterfactuals for Concept Learning in the Semantic Web. In: Ali, M., Esposito, F. (eds.) IEA/AIE 2005. LNCS (LNAI), vol. 3533, pp. 370–379. Springer, Heidelberg (2005)
14. Fanizzi, N., d’Amato, C., Esposito, F.: DL-FOIL Concept Learning in Description Logics. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 107–121. Springer, Heidelberg (2008)
15. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. *Machine Learning* 78(1), 203–250 (2010)
16. Cormen, T.: *Introduction to algorithms*. The MIT Press (2001)
17. Lehmann, J., Auer, S., Tramp, S., et al.: Class expression learning for ontology engineering. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2011)
18. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web* 5(2), 51–53 (2007)
19. Fahnrich, K., Lehmann, J., Hellmann, S.: Comparison of concept learning algorithms (2008)
20. Železný, F., Srinivasan, A., Page, D.L.: Lattice-Search Runtime Distributions May Be Heavy-Tailed. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 333–345. Springer, Heidelberg (2003)

Rule-Based High-Level Situation Recognition from Incomplete Tracking Data

David Münch¹, Joris IJsselmuiden¹, Ann-Kristin Grossefinger¹,
Michael Arens¹, and Rainer Stiefelhagen^{1,2}

¹ Fraunhofer IOSB, Germany


{david.muench, joris.ijsselmuiden, ann-kristin.grossefinger,
michael.arens}@iosb.fraunhofer.de

² Karlsruhe Institute of Technology, Germany
rainer.stiefelhagen@kit.edu

Abstract. Fuzzy metric temporal logic (FMTL) and situation graph trees (SGTs) have been shown to be promising tools in high-level situation recognition. They generate semantic descriptions from numeric perceptual data. FMTL and SGTs allow for sophisticated and universally applicable rule-based expert systems. Dealing with incomplete data is still a challenging task for rule-based systems. The FMTL/SGT system is extended by interpolation and hallucination to become capable of incomplete data. Therefore, one analysis to the robustness of the FMTL/SGT system in situation recognition is removing parts of the ground truth input tracks. The recognition results are compared to ground truth for situations such as “load object into car”. The results show that the presented approach is robust against incomplete data. The contribution of this work is, first, an extension to the FMTL/SGT system to handle incomplete data via interpolation and hallucination, second, a knowledge base for recognizing vehicle-centered situations.

Keywords: rule-based expert system, fuzzy metric temporal logic, situation graph trees, semantic video understanding.

1 Introduction

High-level situation recognition is the process of generating semantic descriptions from a scene observed through machine perception. First, video data needs to be processed by computer vision to obtain corresponding tracks for people, vehicles, and other objects of interest. Second, these tracks need to be processed by high-level situation recognition to detect the occurrence of interesting situations. For this contribution, we concentrate on deducing high-level situations as “loading an object into a car” from tracking data in a surveillance context, as e.g. Figure  depicts.

High-level situation recognition should be able to handle any form of uncertainty. This can be partial knowledge of the current state of the world, phenomena which are not observed by our model, and noisy observations. One kind



Fig. 1. Two scenes from the VIRAT video dataset [10]. Car park *VIRAT_S_000002* (left) and car park *VIRAT_S_000200* (right). Typical situations in this context are getting into or getting out of a vehicle and loading or unloading an object.

of noisy observations is incomplete data from machine perception, in this case video-based tracking. Data gaps can occur when objects are occluded, when objects move through areas without sensor coverage, or when machine perception experiences technical problems. For this contribution, existing methods were extended to handle incomplete data.

This article is structured as follows: Section 2 provides a short overview of related work in high-level situation recognition. Our own approach, the methodological improvements to handle incomplete data, and the particular SGT and FMTL rules for a prototypical surveillance scenario are presented in Section 3. Section 4 describes the evaluation and results. Section 5 provides a conclusion.

2 Related Work

A broad overview in situation recognition is given in the survey papers [15, 11]. The whole field can be roughly divided into two main architectural strategies. On the one hand, there are direct approaches working directly on videos. On the other hand, there are the hierarchical approaches built of several layers. The basic idea of using several layers is splitting up the whole recognition process into specialized recognition methods. Usually, there are some methods performing object detection and tracking, others are combining the gathered information in a temporally and spatially limited context, and finally upon this information the high-level situation recognition is performed. Hierarchical approaches are divided into statistical methods often based on probabilistic graphical models such as Bayesian networks or Markov models, syntactic approaches representing actions through symbols and combining them to situations with grammar-like structures, and description-based approaches using formal languages such as logic to describe situations. Usually, the latter rely on temporal and spatial properties to describe situations [4].

SGTs were presented as knowledge representation for situation recognition based on FMTL in [9]. [6, 7] extended the situation recognition framework to concurrent multi-hypothesis inference and optimized the runtime performance for real-time operation in several domains.

3 Methods

The general framework that underlies the high-level situation recognition is the layered model for cognitive vision systems initially described in [8]. FMTL is a powerful logic which can deal with notions of fuzziness and time. Handling fuzziness allows for handling both uncertainty and inherently vague concepts in the inference process itself. In [2] FMTL and SGTs are applied to the traffic domain and [3] applies them to human behavior. The advantages of using SGTs are the integrated modeling of knowledge, defining the rules and the inference algorithm in a precise formalism and the consolidation in one powerful framework – the SGT-Editor. The internal representation of SGTs is in FMTL rules. The inference algorithm for SGTs is programmed in FMTL, too. Thus, the whole situation recognition is built upon formal FMTL. This allows precise and fast inference about complex information of a particular scene.

3.1 Handling Incomplete Data

Interpolation of Input Data For a deduction at time t , data $x(t)$ from interval $[t - \Delta t_1, t + \Delta t_2]$ is used. Δt_1 and Δt_2 are dependent on the temporal range of the applied FMTL rules. If data is missing from t to $t + \Delta t_0$, each $x(t')$ with $t' \in [t, t + \Delta t_0]$ should be calculated as the average over all corresponding values in $T_p = [t - \Delta t_1, t - 1]$ and $T_f = [t + \Delta t_0 + 1, t + \Delta t_0 + \Delta t_2]$ with Δt_1 and Δt_2 chosen freely:

$$x(t') = \frac{1}{|T_p| + |T_f|} \left(\sum_{t_p \in T_p} (w_{x(t_p)} \cdot x(t_p)) + \sum_{t_f \in T_f} (w_{x(t_f)} \cdot x(t_f)) \right). \quad (1)$$

The weights $w_{x(t_p)}$ and $w_{x(t_f)}$ can be used to reflect a larger influence of T_p or T_f when t' is closer to the beginning or the end of $[t, t + \Delta t_0]$ respectively. This procedure works well for linear metric values, and radial values need to be handled differently with radial metrics.

Hallucinating High-Level Evidence. When rule-based systems are getting more complex they consequently have to deal with increasing challenges of noisy and incomplete input data. The general drawbacks of such a system are when trying to instantiate the preconditions of any situation scheme and all of the rules can be satisfied except of a very few ones which leads to a discontinuation of the situation recognition of the current path of inference.

To overcome this drawback we extended the situation recognition inference algorithm presented in [7] to hallucinate missing evidence. If the predicted situation scheme cannot be instantiated due to missing evidence, the missing evidence is hallucinated. That means, the algorithm creates satisfied dummy predicates for the missing evidence so that the situation scheme can be instantiated. It is, of course, internally known which situation schemes are hallucinated. And finally, the situation graph traversal gets continued with the new hallucinated situation scheme.

3.2 Knowledge Base

Figure 2 depicts the SGT representing the knowledge for the situation recognition. The right specialization edge emerging from *Root* is not included in this paper since it specializes for situations that are not evaluated in Section 4 like people approaching each other, standing together, and walking together.

For a detailed description on traversing the SGT to recognize situations refer to [7]. The traversal starts in the *Root* situation scheme and continues along the left specialization edge emerging from *Root*. Then, the start situation scheme *PatientCar* instantiates a car as the patient for the current agent. From there, the situation schemes *CarFar* and *CarNear* can be reached through temporal edges and so on.

The head of an FMTL rules activated by the SGT in Figure 2 is e.g. *HaveDistance(agent, patient, category)*. The body of this rule consists of $DistanceIs(agent, patient, distance) \wedge AssociateDistance(distance, category)$. $DistanceIs(agent, patient, distance)$ calculates the Euclidian distance which is then associated with distance categories using $AssociateDistance(distance, category)$ as described e.g. in Figure 7 in [2].

4 Evaluation

Experimental Setup. We implemented and evaluated the proposed method on videos with annotated ground truth data from the VIRAT video dataset, see Figure 1. The VIRAT Video Dataset Release 1.0 was made publicly available in 2011 and is presented in [10]. For three out of six places there exist ground truth annotated files where each object or person of interest is annotated. Additionally, in a second file there are annotated semantically interesting situations and all the participating agents in the environment of a car park. The annotated vehicle-centered situations comprising persons, objects and cars are getting into or getting out of a vehicle, opening or closing trunk, and loading or unloading an object of a vehicle.

The provided ground truth annotated data of the VIRAT video dataset is regarded as complete information. In this evaluation every experiment was performed ten times with a probabilistic unique removal of data.

First, we extended the situation recognition system as mentioned in Section 3.1 to be capable of incomplete data. Second, we developed the lower level basic knowledge which is represented in FMTL rules. This universally valid knowledge is domain independent and does not need to be changed when a different domain is considered. Third, the knowledge about the expected situations is encoded in an SGT. Consequently, the specific SGT, see Figure 2, describes all of the expected situations of a certain domain.

Results. We choose the following six videos to evaluate on, due to the availability of annotations and the occurrence of different situations. From scene 00 we selected sequence 02 (a), 03 (b), 04 (c), and 06 (d); from scene 02 we selected segment 06 (e) of sequence 00 and segment 00 (f) of sequence 02.

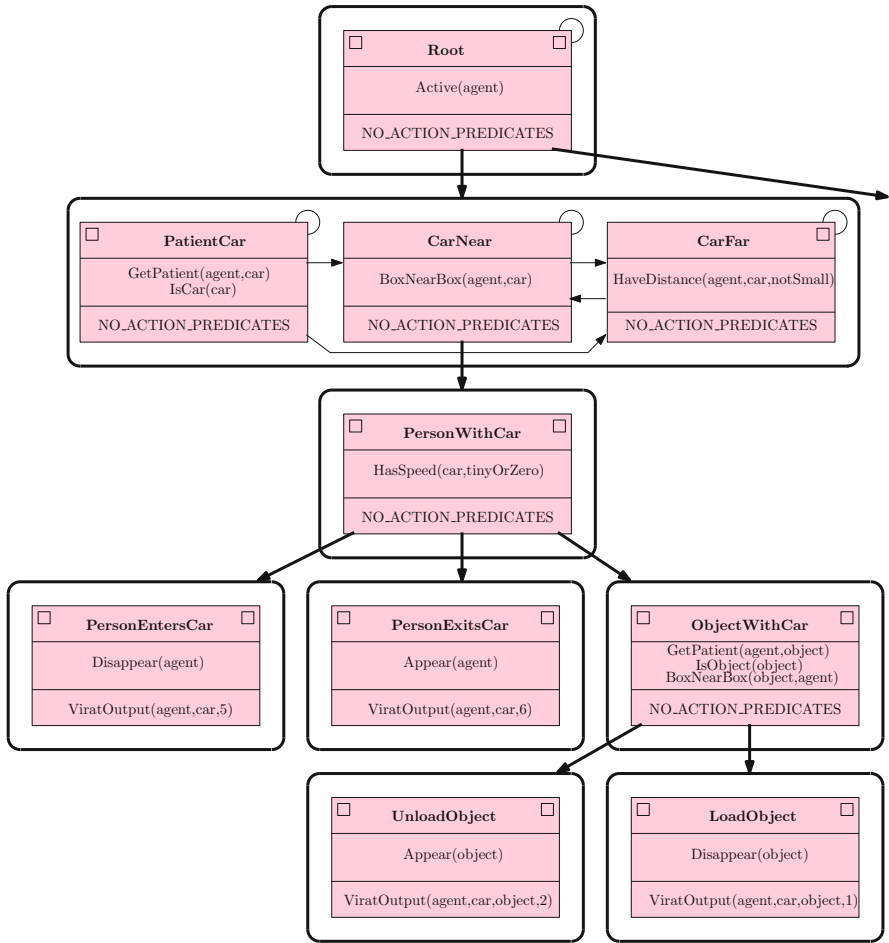


Fig. 2. Part of the SGT representing the knowledge to detect the expected vehicle-centered situations used in the evaluation. The basic structural element of an SGT is a situation scheme which is identified by a unique name, a precondition, and a postcondition both out of one or more FMTL predicates. An example is the “Root” situation scheme with the precondition $Active(agent)$ and without any postcondition. A situation scheme can be a start resp. end situation which is marked with a small box on the upper left resp. right of the situation scheme. Thin edges represent the temporal structure of the situation schemes within a unit visualized with a thick box called situation graph. Thick edges from a single situation scheme to a situation graph model the conceptual refinement of a situation scheme. The resulting structure is a hypergraph and is called SGT. In this figure the situation schemes $PersonEntersCar$, $PersonExitsCar$, $UnloadObject$, and $LoadObject$ raise as postcondition a message that they could be instantiated with a distinct configuration of the variables.

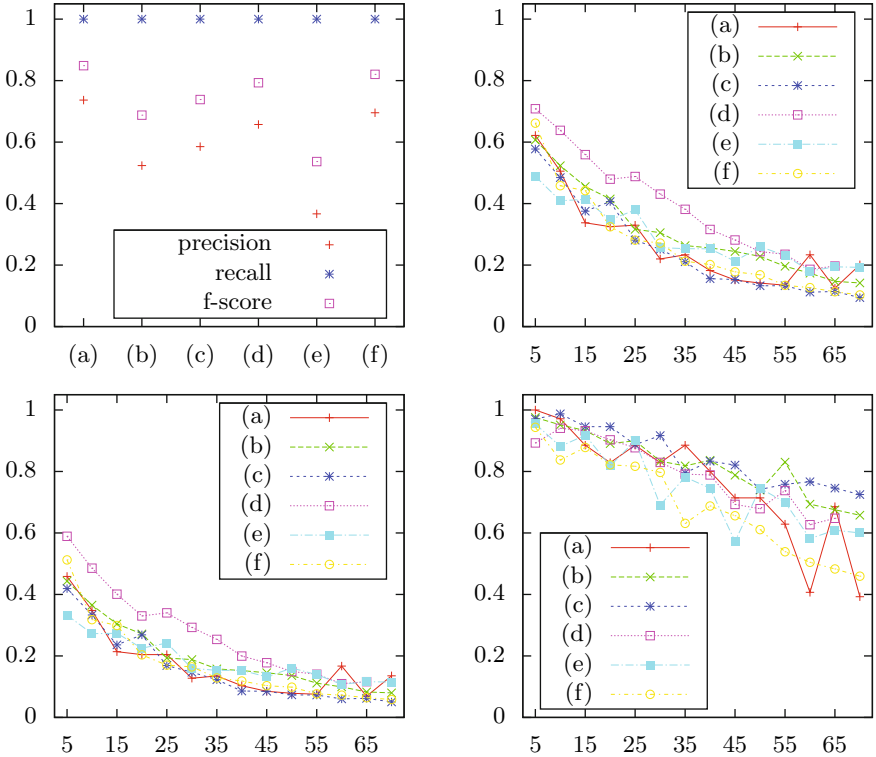


Fig. 3. The results of the unmodified, original scenarios of the six different videos of the VIRAT dataset are visualized in terms of precision, recall, and f-score (upper left). F-score of all evaluated videos with gap size of 5 seconds (upper right), precision (lower left), and recall (lower right). The horizontal axis equals the removed data in percent.

The classification rates of the performed experiments on the six different video sequences are shown in Figure 3 (upper left). The recall is throughout all the six sequences equal to 1.0, which means, that the proposed method never misses any interesting situation in the testset. The average precision is far from 1, the f-score, of course, is slightly better. Some false positive classification results cause the bad precision, but we argue that this is not as disappointing because every single occurring situation was recognized.

Figure 3 depicts f-score (upper right), precision (lower left), and recall (lower right) of all evaluated videos. The figures show that the proposed approach is capable of handling incomplete data even if more than half of the data is missing.

Figure 4 shows the ROC-curves of video (d) for a gap size of 5 seconds (left). The false positive rate slightly increases for larger amounts of missing data and the larger the gaps, the true positive rate decreases slightly. The same evaluation without data interpolation and hallucinating performs worse (right).

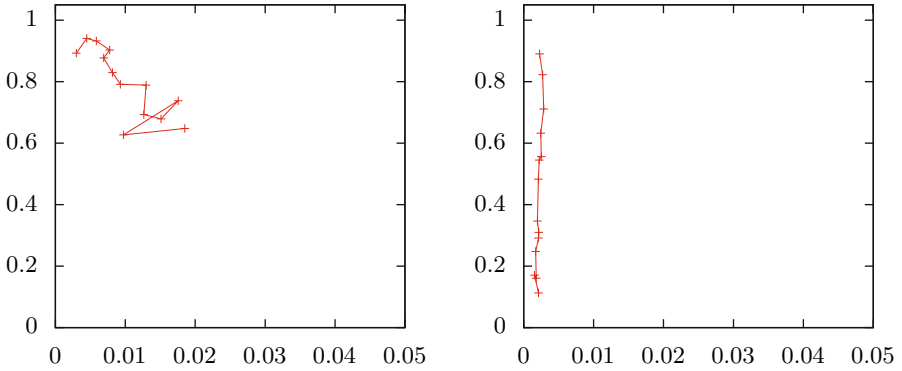


Fig. 4. For video (d) with gap size five seconds the ROC-curves are shown (left). True positive rate on vertical axis; false positive rate on horizontal axis. Without data interpolation and hallucinating (right).

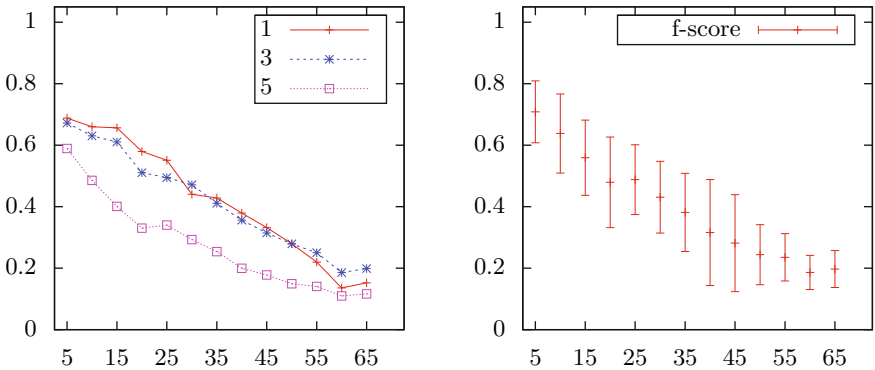


Fig. 5. Video (d) with gap sizes of 1, 3, and 5 seconds (left). F-score (right) of video (d) with gap size 5 including the error bars of the three standard deviations.

Figure 5 (left) consists of three different test configurations: gap sizes of 1, 3, and 5 seconds in video (d). Gap sizes of 1 and 3 perform slightly similar; larger gaps of size 5 result in a roughly worse result. Figure 5 (right) shows f-score of video (d) with gap size 5 including the error bars of the three standard deviations.

5 Conclusion

We have presented a cognitive vision system that can deal with incomplete data in the application of situation recognition in a video surveillance setup. The main ideas to deal with incomplete data in a rule-based expert system are on the lower

tier the interpolation of input data and its uncertainty and on the upper tier the extension of the situation recognition inference algorithm. These two extensions allows our system both to deal with ordinary incomplete data and to handle *high-level incomplete data* such as occlusions. The contribution of this work is the extension of the SGT-Editor and the formal situation recognition inference algorithm to handle incomplete data. As well as developing a knowledge base for recognizing vehicle-centered situations and the broad evaluation of the VIRAT video dataset on a high semantic level. To the best of our knowledge nobody has evaluated the VIRAT video dataset on a high semantic level before.

Acknowledgements. The authors would like to thank Yvonne Fischer and Wolfgang Hübner for fruitful discussions and for their contributions leading to the success of this work.

References

1. Aggarwal, J.K., Ryoo, M.S.: Human Activity Analysis: A Review. *ACM Computing Surveys* (2011)
2. Gerber, R., Nagel, H.-H.: Representation of occurrences for road vehicle traffic. *Artificial Intelligence* 172(4-5), 351–391 (2008)
3. González, J., Rowe, D., Varona, J., Roca, F.X.: Understanding dynamic scenes based on human sequence evaluation. *Image and Vision Computing* 27(10), 1433–1444 (2009); Special Section: Computer Vision Methods for Ambient Intelligence
4. Ijsselmuiden, J., Stiefelhagen, R.: Towards High-Level Human Activity Recognition through Computer Vision and Temporal Logic. In: Dillmann, R., Beyerer, J., Hanebeck, U.D., Schultz, T. (eds.) *KI 2010. LNCS*, vol. 6359, pp. 426–435. Springer, Heidelberg (2010)
5. Lavee, G., Rivlin, E., Rudzsky, M.: Understanding video events: A survey of methods for automatic interpretation of semantic occurrences in video. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* 39(5), 489–504 (2009)
6. Münch, D., Ijsselmuiden, J., Arens, M., Stiefelhagen, R.: High-level situation recognition using fuzzy metric temporal logic, case studies in surveillance and smart environments. In: *ICCV Workshops*, pp. 882–889 (2011)
7. Münch, D., Jüngling, K., Arens, M.: Towards a Multi-purpose Monocular Vision-based High-Level Situation Awareness System. In: *International Workshop on Behaviour Analysis and Video Understanding, ICVS 2011*, p. 10 (2011)
8. Nagel, H.H.: Image sequence evaluation: 30 years and still going strong. In: *International Conference on Pattern Recognition*, vol. 1, p. 1149 (2000)
9. Nagel, H.H.: Steps toward a cognitive vision system. *AI Magazine* 25(2), 31–50 (2004)
10. Oh, S., et al.: A large-scale benchmark dataset for event recognition in surveillance video. In: *CVPR*, pp. 3153–3160 (2011)
11. Turaga, P., Chellappa, R., Subrahmanian, V.S., Udrea, O.: Machine recognition of human activities: A survey. *CSVT* 18(11), 1473–1488 (2008)

Author Index

- Albert, Patrick 17
Al Manir, Mohammad Sadnan 280
Almendros-Jiménez, Jesús M. 248
Amirat, Yacine 120
Arens, Michael 317
Athán, Tara 100, 289
- Bajwa, Imran Sarwar 92
Baker, Christopher J.O. 280
Bassiliades, Nick 193, 215
Batsakis, Sotiris 240
Bauquel, Philippe 17
Berstel-Da Silva, Bruno 47
Boley, Harold 100, 264, 280
Bragaglia, Stefano 151
- Cabot, Jordi 17
Chesani, Federico 151
Chibani, Abdelghani 120
Chniti, Amina 62
Citeau, Hugues 62
Cosentino, Valerio 17
Costantini, Stefania 167
- De Gasperis, Giovanni 167
Demey, Yan Tang 224
Dietrich, Jens 302
- El Ghali, Adil 62
- Fayolle, Jacques 208
- Governatori, Guido 32
Gravier, Christophe 208
Grosselfinger, Ann-Kristin 317
Guesgen, Hans W. 302
Guissé, Abdooulaye 77
- Hashmi, Mustafa 32
- IJsselmuiden, Joris 317
- Jailly, Benjamin 208
- Kowalski, Robert 1
Kravari, Kalliopi 193
Ksysstra, Katerina 136
- Lévy, François 77
- Malik, Saleem 92
Marsland, Stephen 302
Mello, Paola 151
Mugnier, Marie-Laure 16
Münch, David 317
- Nazarenko, Adeline 77
- Pankowski, Tadeusz 256
Paschke, Adrian 100, 182
Patkos, Theodore 120
Perronnet, Jacques 17
Peter-Paul, Reuben 264
Petrakis, Euripides G.M. 240
Plexousakis, Dimitris 120
Preda, Marius 208
- Riazanov, Alexandre 264, 280
- Sadri, Fariba 1
Sottara, Davide 151
Stefaneas, Petros 136
Stiefelhagen, Rainer 317
Subercaze, Julien 208
- Teymourian, Kia 100
Tran, An C. 302
Tran, Trung-Kien 224
Triantafyllou, Nikolaos 136
Tsadiras, Athanasios 215
- Viktoratos, Iosif 215
- Weigand, Hans 182
Wynn, Moe Thandar 32
- Zhao, Zhili 100
Zou, Gen 264