

Texts & Monographs in Symbolic Computation

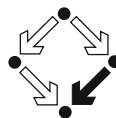
Bernhard Thalheim
Klaus-Dieter Schewe
Andreas Prinz
Bruno Buchberger *Editors*

Correct Software in Web Applications and Web Services

 Springer

Texts and Monographs in Symbolic Computation

A Series of the Research Institute
for Symbolic Computation,
Johannes Kepler University, Linz, Austria



Series Editor: Peter Paule, RISC Linz, Austria

Founding Editor: B. Buchberger, RISC Linz, Austria

Editorial Board

Robert Corless, University of Western Ontario, Canada

Hoon Hong, North Carolina State University, USA

Tetsuo Ida, University of Tsukuba, Japan

Martin Kreuzer, Universität Passau, Germany

Bruno Salvy, INRIA Rocquencourt, France

Dongming Wang, Université Pierre et Marie Curie – CNRS, France

More information about this series at
<http://www.springer.com/series/3073>

Bernhard Thalheim • Klaus-Dieter Schewe •
Andreas Prinz • Bruno Buchberger
Editors

Correct Software in Web Applications and Web Services

 Springer

Editors

Bernhard Thalheim
Institut für Informatik
Christian-Albrechts-Universität
Kiel
Germany

Klaus-Dieter Schewe
Software Competence Center
Hagenberg
Austria

Andreas Prinz
ICT Department
University of Agder
Kristiansand
Norway

Bruno Buchberger
RISC
Johannes Kepler University
Hagenberg
Austria

ISSN 0943-853X ISSN 2197-8409 (electronic)
Texts and Monographs in Symbolic Computation
ISBN 978-3-319-17111-1 ISBN 978-3-319-17112-8 (eBook)
DOI 10.1007/978-3-319-17112-8

Library of Congress Control Number: 2015940569

Springer Cham Heidelberg New York Dordrecht London
© Springer International Publishing Switzerland 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, 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.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media
(www.springer.com)

Preface

This volume constitutes selected and extended papers of a European Science Foundation (ESF) strategic workshop.

The workshop “Correct Software in Web Applications” was held at the Research Institute for Symbolic Computation (RISC), part of the Johannes Kepler University Linz in Hagenberg, Austria, over 3 days. Originally, 30 participants from 12 countries have been invited in September 2011. Twenty-six researchers from Austria, Germany, Great Britain, Hungary, Italy, France, Norway and Romania participated in the workshop. Fourteen researchers were invited for submission of an extended research paper. Finally, we selected nine from these papers for this volume.

The workshop programme consisted of presentations and intensive discussion rounds. The presentations covered abstract state machines (ASMs) and Event-B as formal methods and experiences in applying them to selected fields of Web applications, Theorema and Karlsruhe Interactive Verifier (KIV) as verification methods and experiences made with these tools for Web applications and various detailed descriptions of facets of the large field of Web applications such as Web information systems with emphasis on storyboarding and media types, recommender systems, common protocols such as Hypertext Transfer Protocol (HTTP), browser technology, scripting technology, security aspects of Web services and Web services orchestration. The three discussion sessions addressed the characterisation of what constitutes a Web application, what does correctness mean in this context and what is a common umbrella for a research agenda in this field.

Web applications have become one of the largest area for the application of software engineering methods. At the beginning, Web applications were only simple information services, which soon developed into large database-backed Web information systems, i.e. data-intensive systems that are accessed and maintained via the World Wide Web. More recently, the area of Web services has emerged, referring to software components on some Web servers that can be accessed by and integrated into other software systems. There is a tendency to extend Web information systems and Web services towards large-scale interoperable systems.

This marks a shift towards computation in the public domain using the Internet as the medium for interaction of software components.

Despite the immense importance of Web applications for software engineering, there is a lack of well-founded development methods. Surprisingly, many Web applications are created in an ad hoc manner without the use of sophisticated formal methods. Quality assurance is to a large extent ignored. This constitutes a barrier to productive software development; in other words, the envisioned and most likely technically possible shift to computation in the public domain will only become reality if the resulting systems are trustworthy with respect to consistency, reliability, performance and security. Thus, there is a need for development methods that lead to provably correct Web application systems.

The world and correspondingly information technology (IT) are continuously changing. Systems become accessible through networks. The corresponding infrastructure exists, emerges and provides all what is necessary for cooperations. Therefore, applications can run on systems that are rented by companies. Cloud systems are based on new paradigms such as software-as-a-service, data-as-a-service and infrastructure-as-a-service. Services systems are currently far more advanced than the collaboration systems in the past. At the same time, the theory and conception of a service are not yet properly developed. High-quality services are needed. But despite the fact that services are already an essential part of nowadays IT infrastructures, there remain significant lacunas in our understanding of what services are and of how they work—services are correct and they deliver completely and only what has been asked for—and of proper development of correct and reliable service infrastructures.

Abstract state machines (ASMs) provide a general method to combine specifications on any desired level of abstraction, ground modelling (requirement capture) techniques and stepwise refinement to executable code, providing the basis for experimental validation and mathematical verification. ASMs have been successfully applied to diverse areas such as specification and verification of the implementation of programming languages (e.g. Prolog2WAM, Occam2Transputer, Java2JVM, C#2CLR) and of chip design, train control systems, the Mondex electronic purse and many more. These success stories involve verification by mathematical proofs as well as proofs by theorem provers (e.g. KIV, Prototype Verification System (PVS), Isabelle) or model checking with justifiable effort.

The key problem addressed by the proposed ESF explorative workshop is that there is almost no connection between the research on these industrially successful formal methods in software engineering and the important area of Web applications. Researchers in symbolic computation, abstract state machines, automated reasoning and verification need input from researchers in Web information systems, Web services, interoperability and service-oriented architectures to tailor their research to the needs of the applications, and researchers in Web applications engineering need support to address the challenging correctness problems.

This volume aims to:

1. Obtain a common understanding of the challenging research questions in Web applications comprising Web information systems, Web services and Web interoperability
2. Obtain a common understanding of verification needs in Web applications
3. Achieve a common understanding of the available rigorous approaches to system development and the cases in which they succeeded
4. Identify how rigorous software engineering methods can be exploited to develop correct Web applications
5. Develop a European scale research agenda comprising theory, methods and tools that would lead to correct Web applications with the potential to realise systems for computation in the public domain
6. Develop a formal model of services and facilities for analysis, control and test of such services

The main results are as follows:

- An identification of correctness problems in Web applications and sketches and how these can be solved by formalised software engineering methods, in particular Theorema and ASMs
- An identification of open problems regarding correctness of Web applications and corresponding research questions that have to be addressed in the context of Theorema and ASMs to solve these problems
- A common understanding regarding the need for assuring correctness and the potential of formalised methods with this regard
- A vision for a research agenda, preferably grouped into project topics, to address the open problems regarding correctness of Web applications
- A proposal for a research agenda and a commented list of open problems

Idir Ait-Sadoune and Yamine Ait-Ameur propose an extension of the Business Process Execution Language (BPEL) on the basis of Event-B semantics for formal modelling of Web services compositions that covers the scope, the fault and the compensation handlers. A resulting methodology properly supports the design of transactional BPEL processes. The proposed approach is illustrated by a case study.

Maria Bergholtz, Birger Andersson and Paul Johannesson develop a model of services that allows to properly specify and to analyse the concept of a service based on an understanding of services as a means for cocreation of value, as a means for abstraction and as a means for distributing rights.

Marian Borek, Kuzman Katkalov, Nina Moebius, Wolfgang Reif, Gerhard Schellhorn and Kurt Stenzel develop a development method for secure service applications that integrates a model-driven approach with formal specification techniques using abstract state machines, refinement to code and verification with the interactive theorem prover KIV.

Károly Bósa, Roxana Holom and Mircea Boris Vleju propose a uniform client-cloud interaction approach by which cloud service owners are able to fully control the usages of their services in the case of each user subscription. The applied

method is able to incorporate the major advantages of the ASMs and of ambient calculus.

Ajantha Dahanayake and Bernhard Thalheim develop a conceptual model for service specification based on a general model framework W^*H that extends the rhetoric frame by Hermagoras of Temnos. The framework separates service concerns such as service as a product, service as an offer, service request, service delivery, service application, service record, service log or archive and service exception.

Harald Lampesberger and Mariam Rady show how monitoring is complementing testing and formal methods for Web and cloud systems. The approach extends service-level agreement based on negotiations between clients and providers and thus supports analysis of correctness of the interaction.

Raffaella Mirandola, Pasqualina Potena, Elvinia Riccobene and Patrizia Scandurra present two approaches to predict and analyse reliability of a Web service based on BPEL from one side and on SCA-ASM from the other side. It is shown that the second approach is more effective in comparison with the first one.

Klaus-Dieter Schewe and Qing Wang propose a theory of services (BDCM²) that captures behaviour, description, contracting, monitoring and mediation based on abstract state services, on service mediators and on service ontology models.

Bernhard Thalheim and Klaus-Dieter Schewe survey the codesign approach for integrated development of structuring, functionality, distribution and interactivity for Web information systems. The specification framework has been applied for design and realisation of large information-intensive e-business, edutainment (e-learning), infotainment and community websites.

Reviewers. We thank our reviewers for their efforts, for their detailed reviews and for the support for their second round of reviewing revised papers:

Yamine Ait Aneur
Birger Anderson
Maria Bergholtz
Marian Borek
Karoly Bosa
Ajantha Dahanayake
Antje Düsterhöft
Roxana Holom
Paul Johannesson
Kuzman Katkalov
Meike Klettke
Frank Kramer
Harald Lampesberger
Hui Ma
Raffaella Mirandola

Nina Möbius
Pascalina Potena
Andreas Prinz
Miriam Rady
Wolfgang Reif
Elvinia Riccobene
Patrizia Scandurra
Gerhard Schellhorn
Klaus-Dieter Schewe
Ove Sörensen
Kurt Stenzel
Bernhard Thalheim
Marina Tropmann
Boris Vleju
Qing Wang

Finally, we thank the Springer team for their help, their support and their patience, especially to Silvia Schilgerius.

Kiel, Germany
Hagenberg, Austria
Kristiansand, Norway
Hagenberg, Austria

Bernhard Thalheim
Klaus-Dieter Schewe
Andreas Prinz
Bruno Buchberger

Contents

Formal Modelling and Verification of Transactional Web Service Composition: A Refinement and Proof Approach with Event-B	1
Idir Ait-Sadoune and Yamine Ait-Ameur	
Towards a Model of Services Based on Cocreation, Abstraction and Rights Distribution	29
Maria Bergholtz, Birger Andersson, and Paul Johannesson	
Integrating a Model-Driven Approach and Formal Verification for the Development of Secure Service Applications	45
Marian Borek, Kuzman Katkalov, Nina Moebius, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel	
A Formal Model of Client-Cloud Interaction	83
Károly Bósa, Roxana-Maria Holom, and Mircea Boris Vleju	
W*H: The Conceptual Model for Services	145
Ajantha Dahanayake and Bernhard Thalheim	
Monitoring of Client-Cloud Interaction	177
Harald Lampesberger and Mariam Rady	
Formal Reliability Models for Web Services	229
Raffaella Mirandola, Pasqualina Potena, Elvinia Riccobene, and Patrizia Scandurra	
What Constitutes a Service on the Web?	257
Klaus-Dieter Schewe and Qing Wang	
Codesign of Web Information Systems	293
Bernhard Thalheim and Klaus-Dieter Schewe	

Contributors

Yamine Ait-Ameur IRIT - ENSEEIHT, Toulouse, France

Idir Ait-Sadoune LRI - CentraleSupélec, Gif-Sur-Yvette, France

Birger Andersson Department of Computer and Systems Sciences, Stockholm University, Kista, Sweden

Maria Bergholtz Department of Computer and Systems Sciences, Stockholm University, Kista, Sweden

Marian Borek Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Károly Bósa Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Roxana Chelemen Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Ajantha Dahanayake Department of Computer Information Science, Prince Sultan University, Riyadh, Kingdom of Saudi Arabia

Paul Johannesson Department of Computer and Systems Sciences, Stockholm University, Kista, Sweden

Kuzman Katkalov Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Harald Lampesberger Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Raffaella Mirandola Politecnico di Milano, Milano, Italy

Nina Moebius Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Pasqualina Potena Università degli Studi di Bergamo, Dalmine (BG), Italy

Mariam Rady Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Wolfgang Reif Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Elvinia Riccobene Università degli Studi di Milano, Crema, Italy

Patrizia Scandurra Università degli Studi di Bergamo, Dalmine (BG), Italy

Gerhard Schellhorn Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Klaus-Dieter Schewe Software Competence Center Hagenberg, Hagenberg, Austria
and

Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Kurt Stenzel Institute for Software and Systems Engineering, Augsburg University, Augsburg, Germany

Bernhard Thalheim Department of Computer Science, Christian Albrechts University Kiel, Kiel, Germany

Mircea Boris Vleju Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Hagenberg, Austria

Qing Wang Research School of Computer Science, The Australian National University, Canberra, ACT, Australia

Formal Modelling and Verification of Transactional Web Service Composition: A Refinement and Proof Approach with Event-B

Idir Ait-Sadoune and Yamine Ait-Ameur

Abstract Several languages for describing Web service compositions, like BPEL (Business Process Execution Language), make use of fault and compensation constructs to handle internal and/or external runtime errors of the composed service. This situation particularly occurs for transactional services. However, the absence of a rigorous definition of these BPEL constructors makes it difficult to correctly define the transactional behaviour of a BPEL process. The definitions of such constructs are usually given by their informal descriptions available in the standards. Our contribution proposes an approach to formally define the semantics of these operators. Thus, this paper presents a new Event-B semantics for formal modelling of Web service compositions that covers the scope, the fault and the compensation handlers introduced by the BPEL language specification. It also proposes a methodology showing how we can use Event-B method to design transactional BPEL processes. The proposed approach is illustrated by a case study.

1 Introduction

Service-oriented architectures (SOA) are increasingly used in various application domains. Indeed, a wide range of services operate on the Web and access different distributed and shared resources like databases, data warehouses and scientific calculation repositories. Moreover, the compositions of such services define complex systems not only in the area of computing but also in various businesses like manufacturing or scheduling. Some of these services performing transactional activities are called transactional Web services. This kind of services must satisfy the relevant properties related to transactional systems (atomicity, consistency, isolation and durability (ACID) properties) traditionally required by

I. Ait-Sadoune (✉)

LRI - CentraleSupélec, 3, rue Joliot-Curie, 91190 Gif-Sur-Yvette, France

e-mail: idir.aitsadoune@supelec.fr

Y. Ait-Ameur

IRIT - ENSEEIHT, Toulouse, France

e-mail: yamine@n7.fr

© Springer International Publishing Switzerland 2015

B. Thalheim et al. (eds.), *Correct Software in Web Applications and Web Services*,

Texts & Monographs in Symbolic Computation,

DOI 10.1007/978-3-319-17112-8_1

database management systems. Although the notion of transactions is well mastered by traditional shared and distributed databases, SOA suffers from a lack of formal semantics of transactional Web services. Indeed, in the current SOA tools, overall business transactions may fail or be cancelled when many ACID transactions are committed.

BPEL (Business Process Execution Language [22]) is considered as a standard of Web service composition specification. It offers a compensation mechanism by providing resources for flexible control of reversal and/or resume activities. To reach this goal, BPEL offers the possibility to define fault handling and compensation in an application-specific manner. BPEL has an XML-based textual representation, and its dynamic semantic description is still informally defined in the standards. Due to the lack of formal semantics of BPEL, ambiguous interpretations remain possible, especially the use of different mechanisms (fault handlers and compensation handlers) to handle transactional behaviour. Such languages do not offer the capability to formally handle the transactional Web service aspects and to ensure their correct behaviour.

Several approaches have proposed formal semantics to the BPEL language using different formal descriptions. Petri nets, transition systems, pi-calculus and process algebra have been set up to model BPEL processes; they are summarised in Sect. 8. In our previous work ([5, 6]), we have defined an Event-B-based [2] semantics for the various elements of data and service descriptions and for simple and structured BPEL activities. In this paper, we propose to extend this work to cover the scope, the fault and the compensation handlers introduced by the BPEL language specification. We also propose a methodology to design a transactional BPEL process by assisting a designer for detecting and verifying a transactional behaviour in a BPEL process.

This paper is structured as follows. The next section presents the Event-B method, and Sect. 3 gives an overview of the BPEL language and the case study used in this paper to illustrate the proposed approach. Sections 4, 5 and 6 present, respectively, the proposed approach for analysing transactional Web services and how Event-B method is used to encode these scope, fault and compensation constructs. Section 7 presents an application of the proposed approach to a case study. Section 8 discusses our approach compared to the state of the art in formal verification of BPEL processes and transactional Web services. Finally, a conclusion and some perspectives are outlined.

2 The Event-B Method

The Event-B method [2] is a recent evolution of the B method [1]. This method is based on the notions of preconditions and post-conditions [16], the weakest precondition [9] and the calculus of substitution [1]. It is a formal method based on first-order logic and set theory.

2.1 Event-B Model

An Event-B model is defined by a set of variables, defined in the VARIABLES clause that evolves thanks to events defined in the EVENTS clause. It encodes a state transition system where the variables represent the state and the events represent the transitions from one state to another.

An Event-B model is made of several components of two kinds: Machines and Contexts. The Machines contain the dynamic parts (states and transitions) of a model, whereas the Contexts contain the static parts (axiomatisation and theories) of a model. A Machine can be refined by another one, and a Context can be extended by another one. Moreover, a Machine can see one or several Contexts (Fig. 1).

The refinement operation [3] offered by Event-B encodes model decomposition. A transition system is decomposed into another transition system with more and more design decisions while moving from an abstract level to a less abstract one. A refined Machine is defined by adding new events, new state variables and a gluing invariant. Each event of the abstract model is refined in the concrete model by adding new information expressing how the new set of variables and the new events evolve.

A Context is defined by a set of clauses (Fig. 2) as follows:

- CONTEXT represents the name of the component that should be unique in a model.
- EXTENDS declares the Context extended by the described Context.

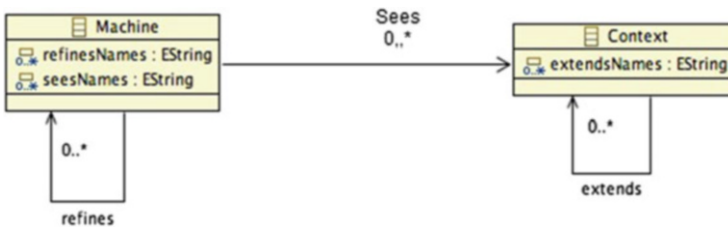


Fig. 1 MACHINE and CONTEXT relationships

CONTEXT	<i>context_identifier₁</i>	MACHINE	<i>machine_identifier₁</i>
EXTENDS	<i>context_identifier₂</i>	REFINES	<i>machine_identifier₂</i>
SETS		SEES	
<i>s</i>		<i>context_identifier₁</i>	
CONSTANTS		VARIABLES	
<i>c</i>		INVARIANTS	
AXIOMS		<i>inv</i> : <i>I(s, c, v)</i>	
<i>axm</i> :	<i>A(s, c)</i>	THEOREMS	
THEOREMS		<i>thm</i> :	<i>T(s, c, v)</i>
<i>thm</i> :	<i>T(s, c)</i>	VARIANT	
END		<i>V(s, c, v)</i>	
		EVENTS	
		< <i>event_list</i> >	
		END	

Fig. 2 The structure of an Event-B development

- SETS describes a set of abstract and enumerated types.
- CONSTANTS represents the constants used by a model.
- AXIOMS describes, in first-order logic expressions, the properties of the attributes defined in the CONSTANTS clause. Types and constraints are described in this clause as well.
- THEOREMS are logical expressions that can be deduced from the axioms.

Similarly to Contexts, a Machine is defined by a set of clauses (Fig. 2). Briefly, the clauses mean:

- MACHINE represents the name of the component that should be unique in a model.
- REFINES declares the Machine refined by the described Machine.
- SEES declares the list of Contexts imported by the described Machine.
- VARIABLES represents the state variables of the model of the specification. Refinement may introduce new variables in order to enrich the described system.
- INVARIANTS describes, by first-order logic expressions, the properties of the variables defined in the VARIABLES clause. Typing information, functional and safety properties are usually described in this clause. These properties shall remain true in the whole model. Invariants need to be preserved by events. It also expresses the gluing invariant required by each refinement for property preservation.
- THEOREMS defines a set of logical expressions that can be deduced from the invariants. They do not need to be proved for each event like for the invariant.
- VARIANT introduces a decreasing natural number which states that the “convergent” events terminate.
- EVENTS defines all the events (transitions) that occur in a given model. Each event is characterised by its guard and by the actions performed when the guard is true. Each Machine must contain an “Initialisation” event. The events occurring in an Event-B model affect the state described in VARIABLES clause.

An event consists of the following clauses (Fig. 3):

- *status* can be “ordinary”, “convergent” (the event has to decrease the variant) or “anticipated” (the event must not increase the variant).
- *refines* declares the list of events refined by the described event.
- *any* lists the parameters of the event.

Fig. 3 Event structure

Event	$evt \hat{=}$
Status	convergent
any	
	x
where	
	$grd : G(s, c, v, x)$
then	
	$act : v : BA(s, c, v, x, v')$
end	

$\langle \text{variable_identifier} \rangle$	$:= \langle \text{expression} \rangle$	(1)
$\langle \text{variable_identifier_list} \rangle$	$: \langle \text{before_after_predicate} \rangle$	(2)
$\langle \text{variable_identifier} \rangle$	$:\in \langle \text{set_expression} \rangle$	(3)

Fig. 4 The kinds of actions of an event

- *where* expresses the guard of the event. An event is fired when its guard evaluates to true. If several guards evaluate to true, only one is fired with a non-deterministic choice.
- *then* contains the actions of the event that are used to modify the state variables.

Event-B offers three kinds of actions that can be deterministic or not (Fig. 4). For the first case, the deterministic action is represented by the “assignment” operator that modifies the value of a variable. This operator is illustrated by the action (1). For the case of the non-deterministic actions, the action (2) represents the “before-after” operator acting on a set of variables whose effect is represented by a predicate, expressing the relationship between the contents of variables before and after the triggering of the action. Finally, the action (3) represents the non-deterministic choice operator, acting on a variable, by modifying its content with an undetermined value in a set of values.

2.2 Proof Obligation Rules

Proof obligations (POs) are associated to any Event-B model. They are automatically generated. The *proof obligation generator plug-in* in the Rodin platform [24] is in charge of generating them. These POs need to be proved in order to ensure the correctness of developments and refinements. The obtained PO can be proved automatically or interactively by the *prover plug-in* in the Rodin platform.

The rules for generating proof obligations follow the substitution calculus [1] close to the weakest precondition calculus [9]. In order to define some proof obligation rules, we use the notations defined in Figs. 2 and 3 where s denotes the seen sets, c the seen constants and v the variables of the Machine. Seen axioms are denoted by $A(s, c)$ and theorems by $T(s, c)$, whereas invariants are denoted by $I(s, c, v)$ and local theorems by $T(s, c, v)$. For an event evt , the guard is denoted by $G(s, c, v, x)$, and the action is denoted by the before-after predicate $BA(s, c, v, x, v')$ (the action (2) of Fig. 4).

- *The theorem proof obligation rule*: this rule ensures that a proposed context or machine theorem is indeed provable:

$$A(s, c) \Rightarrow T(s, c)$$

$$A(s, c) \wedge I(s, c, v) \Rightarrow T(s, c, v)$$

- *Invariant preservation proof obligation rule*: this rule ensures that each invariant in a machine is preserved by each event:

$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \Rightarrow I(s, c, v')$$

- *Feasibility proof obligation rule*: the purpose of this proof obligation is to ensure that a non-deterministic action is feasible:

$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \Rightarrow \exists v'. BA(s, c, v, x, v')$$

- *The numeric variant proof obligation rule*: this rule ensures that under the guards of each convergent or anticipated event, a proposed numeric variant is indeed a natural number:

$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \Rightarrow V(s, c, v) \in \mathbb{N}$$

- *The variant proof obligation rule*: this rule ensures that each convergent event decreases the proposed numeric variant. It also ensures that each anticipated event does not increase the proposed numeric variant. The rule in the case of a convergent event is

$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \Rightarrow V(s, c, v') < V(s, c, v)$$

The rule in the case of an anticipated event is

$$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \Rightarrow V(s, c, v') \leq V(s, c, v)$$

There are other rules for generating proof obligations to prove the correctness of refinement. These rules are given in [2].

2.3 Semantics of Event-B Models

The new aspect of the Event-B method [2], in comparison with classical B [1], is related to the semantics. Indeed, the events of a model are atomic events of state transition systems. The semantics of an Event-B model is trace-based semantics with interleaving. A system is characterised by the set of licit traces corresponding to the fired events of the model which respects the described properties. The traces define a sequence of states that may be observed by properties. All the properties will be expressed on these traces.

This approach proved the capability to represent event-based systems like railway systems, embedded systems or Web services. Moreover, decomposition (thanks to refinement) supports building of complex systems gradually in an incremental manner by preserving the initial properties, thanks to the preservation of a gluing invariant.

3 Service Composition Description Languages

The Web service composition description languages are XML-based languages. The most popular languages are BPEL [22], CDL [30], OWL-S [29], BPMN [23] and XPD [31]. If these languages are different from the description point of view, they share several concepts, in particular the service composition. Among the shared concepts, we find the notions of *activity* for producing and consuming messages; *attributes*, for instance, correlation; message decomposition; service location; compensation in case of failure; events; and event handling. These elements are essential to describe service compositions and their behaviour.

However, due to their XML-based definition, these languages suffer from a lack of semantics. It is usually informally expressed in the standards that describe these languages using natural language or semiformal notations. Hence, the need of a formal semantics expressing this semantics emerged. Formal description techniques and the corresponding verification procedures are very good candidates for expressing the semantics and for verifying the relevant properties.

The formal approach we develop in this paper uses BPEL as a service composition description language and Event-B as a formal description technique. The proposed approach can be extended to be used with other service composition description language.

3.1 Overview of BPEL

BPEL (Business Process Execution Language [22]) is a standardised language for specifying the behaviour of a business process based on interactions between a process and its service partners (*partnerLink*). It defines how multiple service interactions, between these partners, are coordinated to achieve a given goal. Each service offered by a partner is described in a WSDL document through a set of *operations* and of handled *messages*.

WSDL (Web Service Description Language [28]) is a standardised language for describing the published interface (input and output parameter types) of the Web service. It provides with the address of the described service, its identity, the operations that can be invoked and the operation parameters and their types. Thus, WSDL describes the function provided by Web service operations. It defines the exchanged messages that envelop the exchanged data and parameters. The composition of such Web services is described in languages, like BPEL, supporting such composition operators.

A BPEL process uses a set of *variables* to represent the messages exchanged between partners. They also represent the state of the business process. The content of these messages is amended by a set of *activities* which represent the process flow. This flow specifies the operations to be performed, their ordering, activation conditions, reactive rules, etc. Figures 5 and 6 show the XML structure of a WSDL description and a BPEL process.

```

<message name="InfosMessage">
  <part name="DebitInfosPart" type="DebitInfos"/>
  <part name="CreditInfosPart" type="CreditInfos"/>
</message>
<message name="ResponseMessage">
  <part name="ResponsePart" type="int"/>
</message>
<message name="DebitMessage">
  <part name="DebitInfosPart" type="DebitInfos"/>
</message>
<message name="CreditMessage">
  <part name="CreditInfosPart" type="CreditInfos"/>
</message>
<portType name="BankTransferPortType">
  <operation name="BankTransferOperation">
    <input message="InfosMessage"/>
    <output message="InfosOut"/>
  </operation>
  <operation name="DebitOperation">
    <input message="DebitMessage"/>
  </operation>
  <operation name="CreditOperation">
    <input message="CreditMessage"/>
  </operation>
</portType>

```

Fig. 5 The XML WSDL (Web Service Description Language) description of *BankTransfer* services

```

<process name="BankTransfer" ...>
  <variables>
    <variable name="TransactionIn" messageType="InfosMessage"/>
    <variable name="TransactionOut" messageType="InfosOut"/>
    <variable name="DebitInfo" messageType="DebitInfosMessage"/>
    <variable name="CreditInfo" messageType="CreditInfosMessage"/>
  </variables>
  <sequence name="BankTransferProcess">
    <receive name="ReceiveTransferInfos" variable="TransactionIn" ... operation="BankTransferOperation"/>
    <assign name="AssignTransferInfos">...</assign>
    <invoke name="InvokeDebit" ... operation="DebitOperation" inputVariable="DebitInfo"/>
    <invoke name="InvokeCredit" ... operation="CreditOperation" inputVariable="CreditInfo"/>
    <assign name="AssignResponse">...</assign>
    <reply name="Reply" variable="TransactionOut" ... operation="BankTransferOperation" />
  </sequence>
</process/>

```

Fig. 6 The XML BPEL description of *BankTransfer* process

BPEL offers two categories of activities: (1) atomic activities representing the primitive operations performed by the process (they are defined by the *invoke*, *receive*, *reply*, *assign*, *terminate*, *wait* and *empty* activities and correspond to basic Web services) and (2) structured activities obtained by composing primitive activities and/or other structured activities using the *sequence*, *if*, *while* and *repeat Until* composition operators that model traditional sequential control constructs. Three other composition operators are defined by the *pick* operator defining a

non-deterministic choice, the *flow* operator defining the concurrent execution and the *scope* operator defining subprocess execution. BPEL also introduces systematic mechanisms for fault handling by defining a set of activities to be executed for handling possible errors anticipated by the Web service composition designer. A compensation handler can be associated to the fault handler; it starts from the erroneous process itself to undo some steps that have already been completed and return the control back at the identified checkpoints.

3.2 A Case Study

We have chosen to illustrate our approach on the pedagogical case study of the *BankTransfer* commonly used to describe transactional Web services (Figs. 5, 6 and 7). This example describes a service processing a bank transfer between two

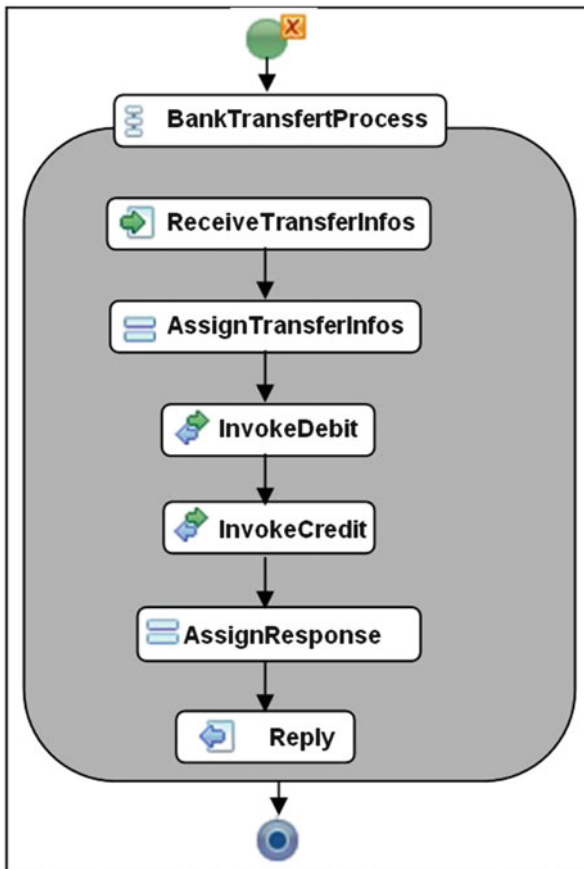


Fig. 7 The graphical BPEL description of *BankTransfer* process

bank accounts from two different banks. On receiving the transfer order from a customer, the process makes, in sequence, a debit from the source bank account and a credit to the target bank account. The BPEL description of Fig. 6 shows how the *BankTransfer* process is decomposed into a sequence of *ReceiveTransferInfos*, *AssignTransferInfos*, *InvokeDebit*, *InvokeCredit*, *AssignResponse* and *Reply* activities. The transaction order information, sent by the customer, is stored in the *TransactionIn* message, and the transaction receipt is sent to the customer in the *TransactionOut* message. The *DebitInfo* and *CreditInfo* messages contain the information sent to the two bank accounts to make the transfer.

4 Event-B for Analysing Transactional Web Services

Our idea is to provide assistance to BPEL developers using the Event-B method [2]. The choice of this formal method is guided by the fact that proof-based methods do not suffer from the state number explosion problem and proofs are eased by the refinement thanks to the decomposition it provides. Our motivation for the use of this formal method is detailed in the previous work [5, 6]. More precisely, our objective is to provide a methodology for detecting the BPEL process parts that handle critical resources. Traditionally, the developer builds its BPEL process and describes the process behaviour without using fault and compensation handlers to guarantee transactional constraints. He/she adds these constraints by observing the behaviour of the obtained process.

In the approach we promote, the designed BPEL process is translated into an Event-B model according to the rules defined in [5] and [6]. Then, the transactional properties and the properties related to the consistencies of resources used by the BPEL process are expressed in the form of consistency invariants (*consistency property*). These invariants are defined by the designer because there is no BPEL resource supporting their expression nor a technique for their checking. Once this enrichment is performed, proof obligations (POs) are generated. Some of these POs related to invariants involving the transactional properties are unprovable because triggering some events separately violates the consistency invariants. Then, BPEL activities related to the event source of these unprovable POs are detected and isolated in a BPEL scope element. This element allows the designer to define a particular BPEL part on which specific mechanisms only apply to this isolated part in an atomic manner. In our case, for transactional BPEL parts, we recommend to apply the mechanisms for fault and compensation handling to the scope element (*compensational atomicity property: guarantees that a transaction consisting of a Web service request will run in such a way that either all of its requests are completed successfully or all requests that occurred during the processing of this transaction will be compensated*) and the “isolated” attribute of the corresponding scope is set to “true”. As a consequence, the execution of this part is isolated by the orchestration tools (*isolation property*), and at the same time, consistency of the resources used by these activities is guaranteed. Transactional properties may

require a redesign of the defined BPEL process. From a methodological point of view, our approach relies on the following steps.

Step 1 Translate the BPEL model into an Event-B model.

Step 2 Introduce, in the Event-B model, the relevant invariants related to the suited transactional behaviour.

Step 3 Isolate the events of the Event-B model whose POs, associated to the introduced invariant of step 2, are not provable.

Step 4 Redesign the BPEL model of step 1 by introducing a BPEL scope embedding the events identified at step 3 and compensation/fault handlers.

This step-based approach is applied until the associated Event-B model is free of unproven POs.

In the following sections, we present the Event-B models for scope, fault handler and compensation handler BPEL concepts. These models extend the proposed ones of the general approach of [5] and [6] to support transactional BPEL processes modelling or checking the behaviour of composed Web services in the case of an internal or external runtime error of a BPEL process.

5 Modelling Scope, Fault and Compensation Handlers

Before addressing the formal modelling of the transactional behaviour, we reviewed the proposed Event-B formal semantics of BPEL ([5, 6]).

5.1 Formal Modelling of BPEL ([5, 6])

Our approach for formal modelling of BPEL with Event-B is based on the observation that a BPEL definition is interpreted as a transition system interpreting the process coordination. A state is represented in both languages by a *variables* element in BPEL and by the VARIABLES clause in Event-B. The various activities of BPEL represent the transitions. They are encoded by events of the Event-B EVENTS clause. For a better understanding of this paper, the transformation rules from BPEL to Event-B models are briefly recalled below. This translation process consists of two parts: *static and dynamic*.

5.1.1 Static Part

The first part translates the WSDL definitions that describe the various Web services and their data types, messages and port types (the profile of supported operations) into the different data types and functions offered by Event-B. This part is encoded in the Context part of an Event-B model.

```

CONTEXT      BankTransferContext
SETS
  Void InfosMessage ResponseMessage DebitMessage CreditMessage DebitInfosType CreditInfosType
CONSTANTS
  DebitInfosPart CreditInfosPart DebitPart CreditPart ResponsePart BankTransferOperation
  DebitOperation CreditOperation
AXIOMS
a12 : DebitInfosPart ∈ InfosMessage → DebitInfosType
a13 : CreditInfosPart ∈ InfosMessage → CreditInfosType
a14 : (DebitInfosPart ⊗ CreditInfosPart) ∈ InfosMessage → (DebitInfosType × CreditInfosType)
a15 : ResponsePart ∈ ResponseMessage → {0, 1}
a16 : DebitPart ∈ DebitMessage → DebitInfosType
a17 : CreditPart ∈ CreditMessage → CreditInfosType
a18 : BankTransferOperation ∈ InfosMessage → ResponseMessage
a19 : DebitOperation ∈ DebitMessage → Void
a20 : CreditOperation ∈ CreditMessage → Void
END

```

Fig. 8 The Event-B CONTEXT of *BankTransfer* process

A BPEL process references data types, messages and operations of the port types declared in the WSDL document. In the following, the rules translating these elements into an Event-B Context are inductively defined:

- 1 The WSDL message element is formalised by an abstract set. Each part attribute of a message is represented by a functional relation corresponding to the template $part \in message \rightarrow type_part$ from the message type to the part type. On the case study in Fig. 5, a set named *InfosMessage* is defined in the SETS clause, and the application of this rule corresponds to the axiom *a12* and *a13* declarations in the *BankTransferContext* in Fig. 8.
- 2 Each operation of a WSDL portType is represented by a functional relation corresponding to the template $operation \in input \rightarrow output$ mapping the input message type on the output message type. On the case study in Fig. 5, the *BankTransferOperation* operation is encoded by the functional relation described by axiom *a18* in Fig. 8.

5.1.2 Dynamic Part

The second part concerns the description of the orchestration process of the activities appearing in a BPEL description. These processes are formalised as Event-B events; each simple activity becomes an event of the Event-B model, and each structured or composed activity is translated to a specific event construction. This part is encoded in a Machine of an Event-B model.

A BPEL process is composed of a set of variables and a set of activities. Each BPEL variable corresponds to a state variable in the VARIABLES clause, and the

MACHINE	BankTransferMachine
SEES	BankTransferContext
VARIABLES	<i>TransactionIn DebitInfo CreditInfo Response varSeq_I</i>
INVARIANTS	<i>i1: TransactionIn \subseteq InfosMessage \wedge card(TransactionIn) \leq 1</i> <i>i2: DebitInfo \subseteq DebitMessage \wedge card(DebitInfo) \leq 1</i> <i>i3: CreditInfo \subseteq CreditMessage \wedge card(CreditInfo) \leq 1</i> <i>i4: Response \subseteq ResponseMessage \wedge card(Response) \leq 1</i> <i>i5: varSeq-I \in {0,1,2,3,4,5,6}</i>

Fig. 9 The Event-B MACHINE of *BankTransfer* process (part 1)

activities are encoded by events. This transformation process is inductively defined on the structure of a BPEL process according to the following rules:

- 3 The BPEL variable element is represented by a variable in the VARIABLES clause in an Event-B Machine. This variable is typed in the INVARIANTS clause using messageType BPEL attribute. The variables and invariants corresponding to the case study are given in Fig. 9. For example, the BPEL variable *DebitInfo* is declared and typed.
- 4 Each BPEL simple activity is represented by a single event in the EVENTS clause of the Event-B Machine. For example, in Fig. 10, the *ReceiveTransferInfos* BPEL atomic activity is encoded by the *ReceiveTransferInfos* Event-B event.
- 5 Each BPEL *structured activity* is modelled by an Event-B description which encodes the carried composition operator. Modelling composition operations in Event-B follow the modelling rules formally defined in [4]. Again, on the same example (Fig. 6), the structured activity *BankTransferProcess* is encoded by a sequence of six events controlled by the *varSeq_I* variable initialised to value 6 (Fig. 10).

When the Event-B models formalising a BPEL description are obtained, they may be enriched by the relevant properties that formalise the user requirements and the soundness of the BPEL-defined process like BPEL type control, orchestration and service composition, deadlock freeness, no live-lock, precondition for calling a service operation and data transformation ([5, 6]).

5.2 An Event-B Model for Scope

The BPEL language provides a particular mechanism for subprocess description thanks to the scope construct. Scope encapsulates subprocess behaviours and includes a context used by the execution of the activities that describe its behaviour. This context contains a state composed by a set of variables. Each scope has a required primary activity that describes its behaviour (Fig. 11).

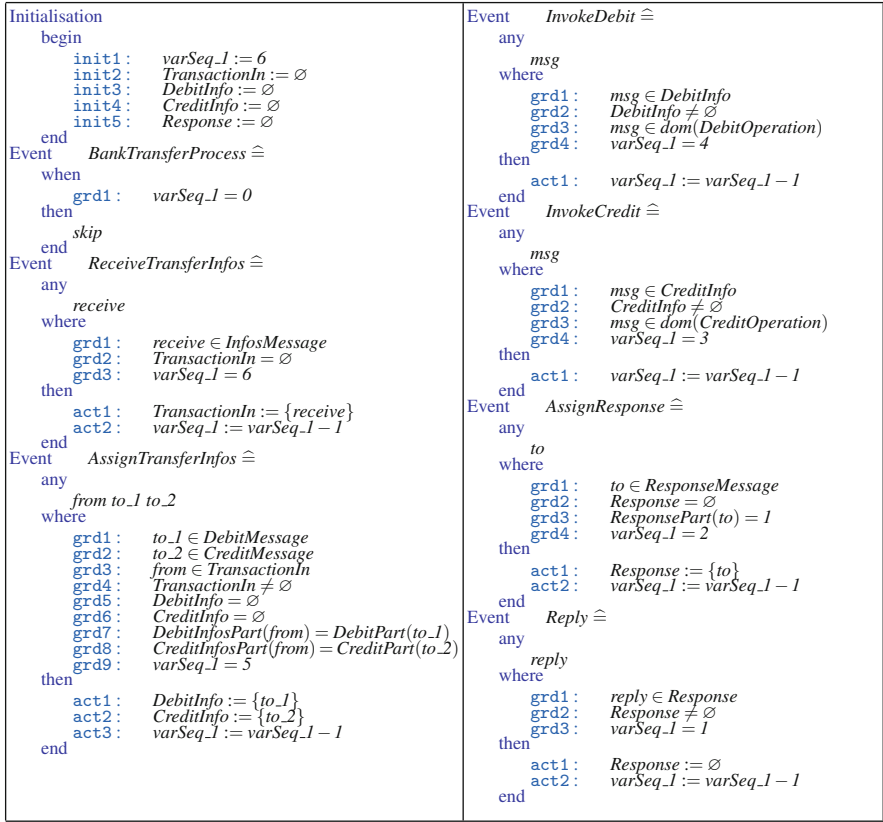


Fig. 10 The Event-B MACHINE of *BankTransfer* process (part 2)



Fig. 11 A BPEL *scope* element containing fault and compensation handlers

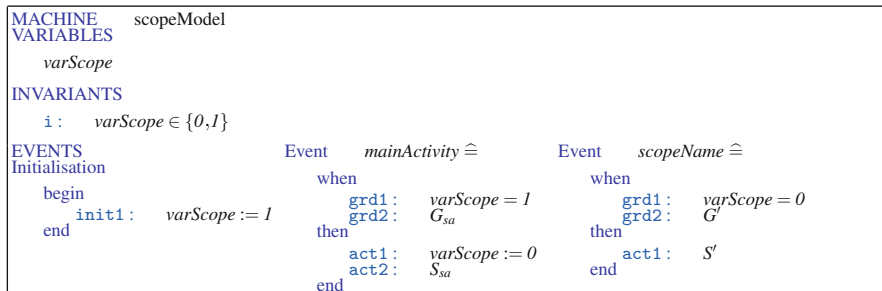


Fig. 12 An Event-B model for a BPEL scope element

The *scopeModel* MACHINE in Fig. 12 formalises the scope behaviour. This MACHINE introduces a *mainActivity* event that models the main activity of a scope and a *scopeName* event that models the end of a scope execution. The variable *varScope* is initialised to 1. This variable triggers the *scopeName* event at the end of the *mainActivity* event. If the scope’s main activity is of structured type, the transformation rules defined for structured activities in [5] are applied. Similarly, the scope variables are modelled according to the variable transformation rules defined in [5].

G_{sa} , G' , S_{sa} and S' represent the guards and the actions of the *mainActivity* and *scopeName* events. They result from variable and activity modelling and are related to the data manipulation and to the process behaviour. The same reasoning applies for fault and compensation handler models presented below.

5.3 An Event-B Model for Fault Handler

A BPEL fault handler is a process that is triggered when an error rose from a partner Web service or an internal BPEL process. It provides a mechanism to define a set of customised “fault-handling” activities. A *catch* element is defined to intercept (to catch) a specific kind of fault, defined by a “*faultName*” attribute. If no predefined name is associated to the intercepted fault, this fault will be processed by a *catchAll* element (see Fig. 11).

The *faultHandlerModel* MACHINE in Fig. 13 formalises a fault Handler as described in Fig. 11. The *mainActivity* event models the normal behaviour of the scope. Different types of errors are taken into account by the defined fault handler. Error types are defined by an enumerated set called *faultType* in the SETS clause. It contains all fault types identified by the designer and used in the BPEL process description. The model in Fig. 13 formalises the case of errors called *fault_1* among other faults. A *varFault* variable is introduced to determine whether the current behaviour of the process is catching errors ($varFault=1$) or describes a normal execution ($varFault=0$).

CONTEXT	faultHandlerContext		
SETS	<i>faultType</i>		
CONSTANTS	<i>fault_1</i> <i>otherFault</i>		
AXIOMS	a1 : <i>partition</i> (<i>faultType</i> , { <i>fault_1</i> }, { <i>otherFault</i> })		
END			
MACHINE	faultHandlerModel		
SEES	faultHandlerContext		
VARIABLES	<i>currentFault</i> <i>varFault</i>		
INVARIANTS	i1 : <i>currentFault</i> ∈ <i>faultType</i> i2 : <i>varFault</i> ∈ {0,1}		
EVENTS	Event	<i>mainActivity</i> ≐	
Initialisation	when	when	
begin	grd1:	grd1:	<i>varFault</i> = 0
init1 : <i>varFault</i> := 0	grd2:	grd2:	<i>G</i>
init2 : <i>currentFault</i> := ∈	then	then	<i>S</i>
<i>faultType</i>	end	end	
end	Event	<i>faultOccurs</i> ≐	
any	when	Event	<i>catchFault_1</i> ≐
where	grd1:	grd1:	<i>varFault</i> = 1
grd1 : <i>varFault</i> = 0	grd2:	grd2:	<i>currentFault</i> =
grd2 : <i>ff</i> ∈ <i>faultType</i>	<i>fault_1</i>	<i>currentFault</i> =	<i>varFault_1</i>
then	grd3 : <i>G₁</i>	<i>otherFault</i>	=
act1 : <i>varFault</i> := 1	then	grd3 : <i>G_n</i>	=
act2 : <i>currentFault</i> := <i>ff</i>	end	then	<i>act1</i> : <i>S_n</i>
end	Event	<i>catchAllFaults</i> ≐	
	where	end	

Fig. 13 An Event-B model for a BPEL fault handler

The occurrence of an error is formalised by the *faultOccurs* event that is arbitrarily triggered (non-deterministic occurrence of the event). When this event is triggered, it ceases the normal behaviour described by the *mainActivity* event thanks to the action *varFault:=1*. The fault processing, corresponding to the error defined by the *currentFault* variable content, starts. If this variable contains the *fault_1* value, the *catchFault_1* event, formalising the processing of the *fault_1* error, is triggered. Otherwise (*currentFault=otherFault*), the *catchAllFaults* event, formalising the *catchAll* element processing, is triggered.

Similarly, this model can be used to formalise a fault handler associated to a BPEL process.

5.4 An Event-B Model for Compensation Handler

A compensation handler is a wrapper for a process that performs compensation. A compensation handler for an activity is available for invocation only when the activity completes successfully. Generally, the invocation of a compensation handler

```

<process ... >
  ...
  <faultHandlers>
    <catch faultName="scopeFault"... >
      <compensate name="catchScope"/>
    </catch>
  </faultHandlers>
  <sequence name="mainActivity">
    ...
    <scope name="scopeName" ...>
      <compensationHandler>
        <activity name="compensationActivity">...</activity>
      <compensationHandler>
    ...
  </scope>
</sequence>
</process>

```

Fig. 14 BPEL fault and compensation handlers

is achieved by a fault handler to undo the effects of already completed activities in the case of a runtime error.

The BPEL process described in Fig. 14 contains two behaviours: the normal behaviour described by the *mainActivity* sequence element and the error catching behaviour described by the fault handler. The normal behaviour consists of a sequence of activities, and one of these activities is required to be a scope. This scope contains a compensation handler to cancel its effect in the case of a runtime error. Then, the fault handling process consists of triggering the *compensate* activity.

The *compensationModel* MACHINE described in Fig. 15 formalises a BPEL process conforming to Fig. 14. The normal behaviour described by a sequence activity is modelled by a *mainActivity* event according to the defined rules in [5]. Being the j th activity of this sequence, the scope is modelled by the *scopeName* event which represents part of the scope model obtained using the rules defined in Sect. 5.2. The *scopeName* event is triggered when $varSeq=j$.

The fault handler part is modelled by the *catchScope* event. It formalises a *compensate* activity named *catchScope*. This part describes the *scopeFault* handling process. The *catchScope* event triggers the compensation handler contained in the *scopeName* scope, in the case that *currentFault* is “scopeFault”. This action is done by assigning value 1 to the *varComp* variable.

The compensation process is described by the *compensationActivity* event. It can be triggered if the *scopeName* scope has completed its execution ($varSeq < j$) and the *catchScope* event is triggered ($varComp=1$). The *compensationActivity* specifies the process of compensation. It can be described by one or a set of activities. In this case, the transformation rules of BPEL activities defined in [5] are applied to detail this description.

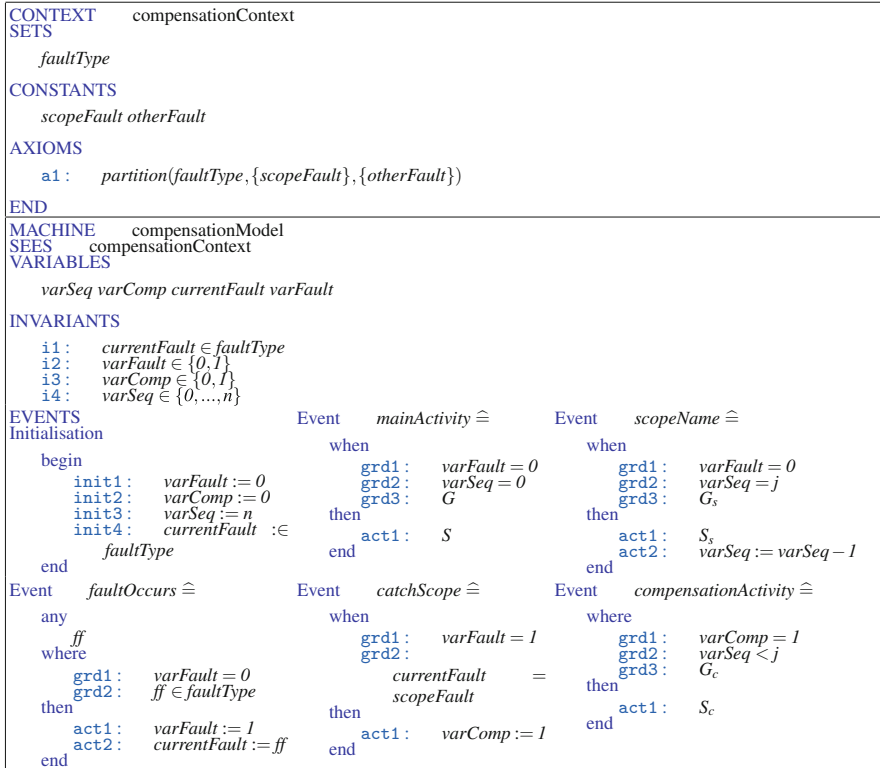


Fig. 15 An Event-B model for BPEL fault and compensation handlers

6 Use of the Tools

The proposed approach is implemented by integrating various plug-ins in the Rodin platform which is based on the Eclipse core. We have used existing plug-ins of Eclipse platform (*BPEL editor*¹) and of Rodin platform (*Event-B editor, prover* [24] and *ProB model checker* [18]) and developed plug-ins (*BPEL2B plug-in* [5]). These plug-ins are used at different steps of our approach:

- Designing BPEL process with the *BPEL editor* and translating the obtained model into an Event-B model using the BPEL2B plug-in (*step 1*).
- Introducing the relevant invariants related to the suited transactional behaviour with the *Event-B editor* (*step 2*). Isolating the events of the model whose POs, associated to the introduced invariant of step 2, cannot be proved within the *Rodin prover*.

¹BPEL Designer Project: <http://eclipse.org/bpel/>

- The *ProB model checker* assisting the developer to confirm the diagnostic and to get a counterexample associated to the isolated events (*step 3*).
- Redesigning the BPEL model of step 1 by introducing a BPEL *scope* embedding the events identified at step 3 and a compensation/fault handler (*step 4*). The graphical view of the *BPEL editor* can be used at this step to facilitate the insertion of the new elements.

7 Application to the Case Study

The application of the approach outlined in Sect. 4 on the example in Fig. 6 is given in the following steps. The reader can find all Event-B CONTEXTs and MACHINES source codes on this website.²

Step 1 This step on the *BankTransfer* BPEL process in Fig. 6 leads to the Event-B model in Figs. 8, 9 and 10 described in Sect. 5.1.

Step 2 The transactional properties are expressed in the form of invariant in the Event-B model of step 1. Starting from the model in Figs. 8 and 10, we obtain the model in Fig. 16 by defining a default fault handler with the *defaultFaultHandler* event. *BankAccount1* and *BankAccount2* variables formalise the contents of two bank accounts and the *amount* variable contains the value to be transferred. The *InvokeDebit* and *InvokeCredit* events invoke Web services that make the transaction. The *consistency* property is expressed by invariant *i12*. The sum of *BankAccount1* and *BankAccount2* variables shall be constant to ensure consistency of the contents of two bank accounts before triggering *InvokeDebit* event and after triggering *InvokeCredit* event. Invariant *i13* expresses the state after triggering *InvokeDebit* event and before triggering the *InvokeCredit* event.

Step 3 The POs are generated by the Rodin platform, and those associated to the invariant *i12* and to the action *varSeq_1:=0* of *defaultFaultHandler* event, which aborts the BPEL process, cannot be proved (Fig. 17). An inconsistency state results from this abortion.

The invariant violation is usually caused by *defaultFaultHandler* event triggering when the *faultOccurs* event is triggered after *InvokeDebit* event. This diagnostic is confirmed by the ProB model checker [18] that gets a counterexample corresponding to this scenario (Fig. 18). The inconsistency state is *varSeq=0* (*defaultFaultHandler* event) and $BankAccount1 + BankAccount2 = ConsistencyState - amount$.

Step 4 If an error occurs during the transaction, a fault handler should restore a consistent state of the process before aborting it. The *InvokeDebit* and *InvokeCredit* activities are isolated in the BPEL scope, a fault handler is associated

²<http://idir.aitsadoune.free.fr>

<pre> CONTEXT BankTransferContext SETS ...faultType CONSTANTS ...ConsistencyState otherFault AXIOMS ...: ... a12: ... partition(faultType, {otherFault}) a13: ... ConsistencyState ∈ ℕ_I END </pre>	
<pre> MACHINE BankTransferWithoutScope SEES BankTransferContext VARIABLES ...varFault currentFault BankAccount1 BankAccount2 amount INVARIANTS ...: ... i7: ... varFault ∈ ℕ ∧ varFault ∈ {0,1} i8: ... currentFault ∈ faultType i9: ... BankAccount1 ∈ ℕ i10: ... BankAccount2 ∈ ℕ i11: ... amount ∈ ℕ i12: ... (varSeq-1 ≠ 3) ⇒ (BankAccount1 + BankAccount2 = ConsistencyState) i13: ... (varSeq-1 = 3) ⇒ (BankAccount1 + BankAccount2 = ConsistencyState - amount) EVENTS Initialisation begin ...: ... init3: ... BankAccount1, BankAccount2 : [... ∧ BankAccount1' + BankAccount2' = Consistency) init4: ... amount : ∈ ℕ_I init5: ... currentFault : ∈ faultType init6: ... varFault := 0 end Event faultOccurs ≐ when grd2: ... varFault = 0 grd3: ... varSeq-1 ≠ 0 then sub1: ... currentFault := otherFault sub2: ... varFault := 1 end Event defaultFaultHandler ≐ when grd1: ... varFault = 1 grd2: ... currentFault = otherFault then sub1: ... varSeq-1 := 0 end </pre>	
<pre> Event ReceiveTransferInfos ≐ Event AssignTransferInfos ≐ Event InvokeDebit ≐ where ...: ... grd4: ... varSeq-1 = 4 grd5: ... amount ≤ BankAccount1 grd6: ... varFault = 0 then act1: ... varSeq-1 := varSeq-1 - 1 act2: ... BankAccount1 := BankAccount1 - amount end Event InvokeCredit ≐ where ...: ... grd4: ... varSeq-1 = 3 grd5: ... varFault = 0 then act1: ... varSeq-1 := varSeq-1 - 1 act2: ... BankAccount2 := BankAccount2 + amount end Event AssignResponse ≐ Event Reply ≐ Event BankTransferProcess ≐ </pre>	

Fig. 16 The consistency property expression of *BankTransfer* process

to this scope, and the mechanism for compensation handling is applied to the *InvokeDebit* and *InvokeCredit* activities (*compensational atomicity property*). The “isolated” attribute of this scope is also set to “true” (*isolation property*). The BPEL process obtained by this step is given in Figs. 19 and 20.

When applying again *step 1* on the obtained BPEL process, the Event-B model in Fig. 21 with *BankTransferContext* and *BankTransferMachine* components is obtained. Only elements that differ from the Event-B model in Figs. 8 and 10 are shown in Fig. 21. The *BankTransferProcess* is encoded by a sequence of five events. One of them is *BankTransferScope* which formalises the *BankTransferScope*

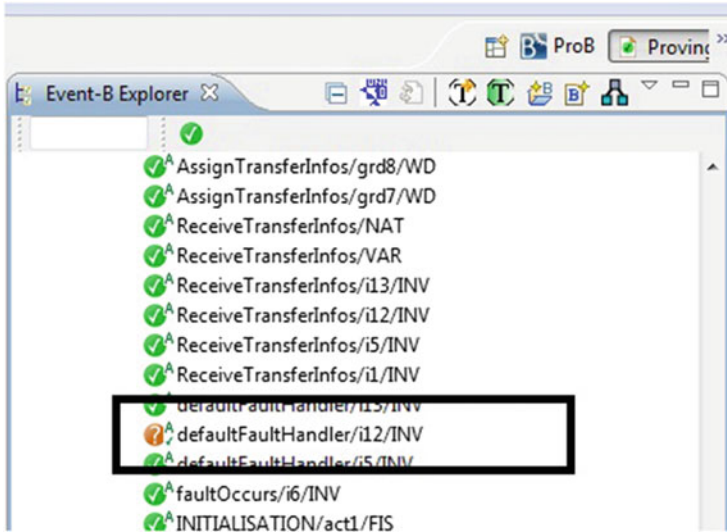


Fig. 17 The Rodin diagnostic

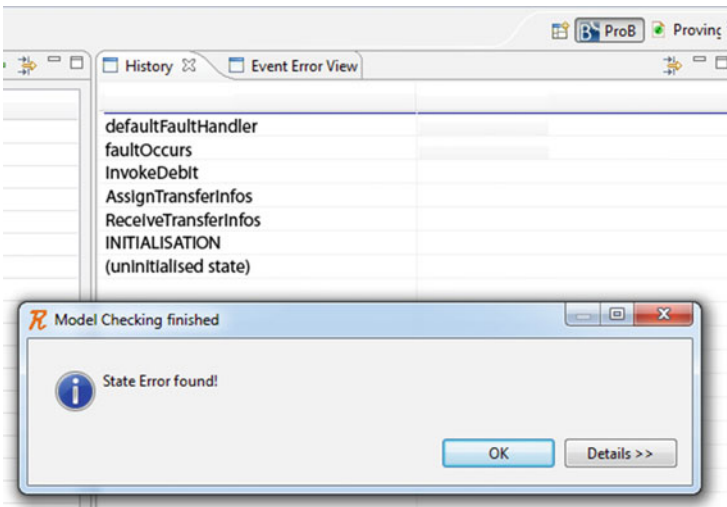


Fig. 18 The ProB diagnostic

scope. The main activity of this scope is a *BankTransfer* activity which is decomposed to a sequence of *InvokeDebit* and *InvokeCredit* activities controlled by the *varSeq_2* variable initialised to value 2. The *scopeFaultOccurs* event formalises the triggering of a runtime error inside the scope. This event triggers the scope fault handler by assigning the *currentFault* variable to “scopeFault”. The fault handler process is described by the *compensate*, *rethrow* and *scopeFaultHandler*


```

<process name="BankTransfer" ...>
...
<sequence name="BankTransferProcess">
  <receive name="ReceiveTransferInfos" ... />
  <assign name="AssignTransferInfos"> ... </assign>
  <scope name="BankTransferscope" isolated="true">
    <faultHandlers>
      <catchAll>
        <sequence>
          <compensate/>
          <rethrow/>
        </sequence>
      </catchAll>
    </faultHandlers>
    <sequence name="BankTransfer">
      <invoke name="InvokeDebit" ...>
        <compensationHandler>
          <invoke name="InvokeCancelDebit" .../>
        </compensationHandler>
      </invoke>
      <invoke name="InvokeCredit" ...>
        <compensationHandler>
          <invoke name="InvokeCancelCredit" .../>
        </compensationHandler>
      </invoke>
    </sequence>
  </scope>
  <assign name="AssignResponse">...</assign>
  <reply name="Reply" .../>
</sequence>
</process/>

```

Fig. 19 The BPEL description of a redesigned *BankTransfer* process

events. The *compensate* event triggers different compensation handlers that consist of invoking *InvokeCancelDebit* and *InvokeCancelCredit* events to cancel the effect of the *InvokeDebit* and *InvokeCredit* events.

In this case, the invariant *i12* is changed and adapted to the modifications made by introducing a scope. All POs are proved, and unlike the case in Fig. 16, the invariant *i12* is not violated by the fault handler process if *scopeFaultOccurs* event is triggered after *InvokeDebit* event (Fig. 22). The *scopeFaultHandler* event triggers the compensation process (*compensate* event) before aborting the BPEL process.

8 Related Work

Various approaches have been proposed to model and to analyse BPEL processes. Most of the work in the literature shows that the proposed approaches use transition systems for representing the business processes, activities and workflows and model checking as the underlying formal verification technique for property validation. *Hinz et al.* [15] and *van der Aalst et al.* [27] have used Petri nets to encode

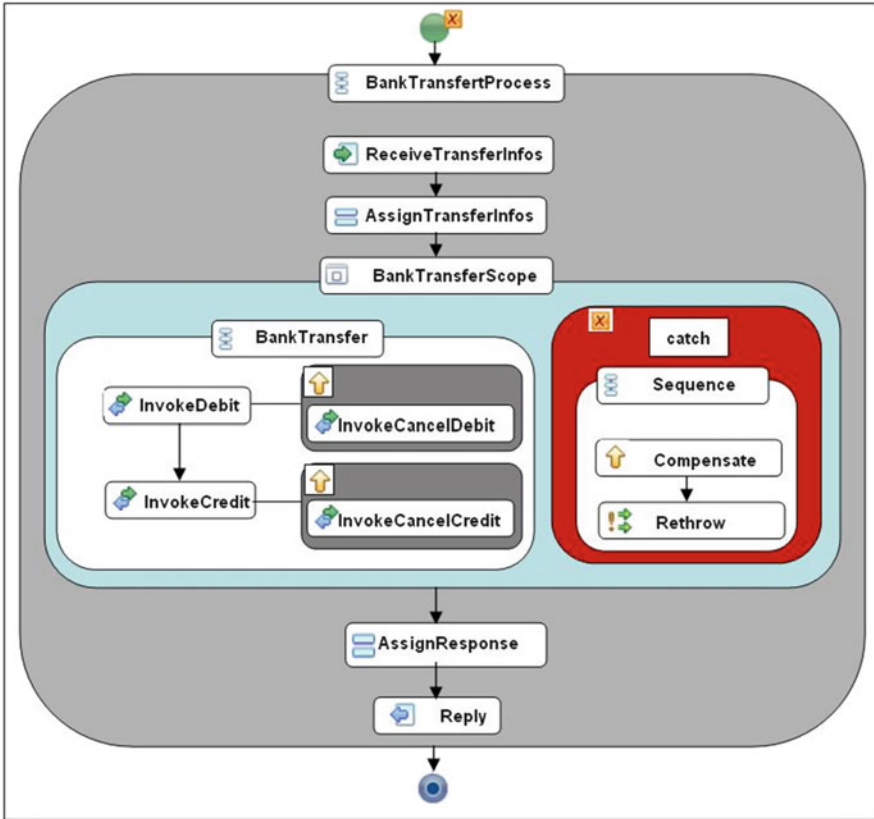
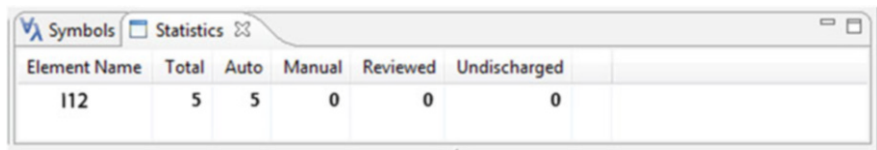


Fig. 20 The graphical BPEL description of a redesigned *BankTransfer* process

BPEL processes. *Nakajima* [21] has mapped a BPEL activity part to a finite automaton encoded in Promela. FSP (finite state process) and the associated tool (LTSA) are used by *Foster et al.* [12] to check if a BPEL Web service composition behaves like an MSC Web service composition specification captured by message sequence charts (MSCs). *Marconi et al.* [20] present the approach that translates a BPEL process to a set of state transition systems. These systems are composed of parallel and the resulting parallel composition is annotated with specific Web service requirements. *Salaun et al.* [25] show how BPEL processes are mapped to processes expressed by LOTOS and CCS process algebra operations. Abstract state machines (ASM) have been developed by *Farahbod et al.* [11] and *Fahland* [10] to model BPEL composition process descriptions. *Borger et al.* [7] have used ASM for modelling workflow, specifically BPMN process. The B method and StaAC were used by *Butler et al.* [8] to model business transactions and their compensation with an application to a BPEL process. Other approaches proposed formal models for Web service compositions. An overview of these approaches can be found in [26].

<pre> CONTEXT BankTransferContext SETS ..faultType CONSTANTS ...otherFault scopeFault CancelDebitOperation CancelCreditOperation AXIOMS ... op1: ... CancelDebitOperation ∈ DebitMessage → Void op2: ... CancelCreditOperation ∈ CreditMessage → Void a13: partition(faultType, {otherFault}, {scopeFault}) END MACHINE BankTransferMachine SETS BankTransferContext VARIABLES ...varSeq_1 varSeq_2 varSeq_3 varScope varComp... INVARIANTS ... i5: ... varSeq_1 ∈ {0, 1, 2, 3, 4, 5} i6: ... varSeq_2 ∈ {0, 1, 2} i8: ... varScope ∈ {0, 1} i9: ... varComp ∈ {0, 1} i10: ... varSeq_3 ∈ {0, 1, 2} ... i12: ... (varSeq_2 ≠ 1) ⇒ (BankAccount1 + BankAccount2 = ConsistencyState) i13: ... (varSeq_2 = 1) ⇒ (BankAccount1 + BankAccount2 = ConsistencyState - amount) EVENTS Initialisation begin init1: varSeq_1, varSeq_2 := 5, 2 init2: varSeq_3, varScope := 2, 1 init3: varFault, varComp := 0, 0 end ... Event otherFaultOccurs ≐ when grd2: varFault = 0 grd3: varSeq_1 ≠ 3 then sub1: currentFault := otherFault sub2: varFault := 1 end Event defaultFaultHandler ≐ when grd1: varFault = 1 grd2: currentFault = otherFault then sub1: varSeq_1 := 0 sub2: varFault := 0 end Event scopeFaultOccurs ≐ when grd2: varFault = 0 grd3: varSeq_1 = 3 then sub1: currentFault := scopeFault sub2: varFault := 1 end Event compensate ≐ when grd1: currentFault = scopeFault grd2: varFault = 1 g3: varSeq_2 = 2 then sub1: varComp := 1 a1: varSeq_3 := varSeq_3 - 1 end Event rethrow ≐ when g1: varFault = 1 g2: varSeq_3 = 1 g3: varSeq_2 = 2 then a1: currentFault := otherFault a2: varSeq_3 := varSeq_3 - 1 end Event ScopeFaultHandler ≐ when g1: varSeq_3 = 0 then skip end </pre>		<pre> Event invokeCancelDebit ≐ any msg where grd1: varComp = 1 grd2: varSeq_2 = 1 grd3: DebitInfo ≠ ∅ grd4: msg ∈ DebitInfo grd5: msg ∈ dom(CancelDebitOperation) then sub1: BankAccount1 := BankAccount1 + amount sub2: varSeq_2 := 2 sub3: varScope := 1 sub4: varComp := 0 end Event invokeCancelCredit ≐ any msg where grd1: varComp = 1 grd2: varSeq_2 < 1 grd3: CreditInfo ≠ ∅ grd4: msg ∈ CreditInfo grd5: msg ∈ dom(CancelCreditOperation) grd6: amount ≤ BankAccount2 then sub1: BankAccount2 := BankAccount2 - amount sub2: varSeq_2 := 1 end Event ReceiveTransferInfos ≐ Event AssignTransferInfos ≐ Event InvokeDebit ≐ Event InvokeCredit ≐ Event BankTransfer ≐ when grd1: varScope = 1 grd2: varSeq_1 = 3 grd3: varSeq_2 = 0 nf: varFault = 0 then sub1: varScope := 0 end Event BankTransferScope ≐ when grd1: varSeq_1 = 3 grd2: varScope = 0 nf: varFault = 0 then act1: varSeq_1 := varSeq_1 - 1 end Event AssignResponse ≐ Event Reply ≐ Event BankTransferProcess ≐ END </pre>
--	--	--

Fig. 21 The Event-B model of the redesigned *BankTransfer* process



Element Name	Total	Auto	Manual	Reviewed	Undischarged
i12	5	5	0	0	0

Fig. 22 The Rodin statistic view about i12 invariant

In all these proposals, a BPEL description is transformed to a formal model to be checked. However, some of this work did not take into account the fault and compensation mechanisms that enable transactional behaviours of BPEL processes. No formal generic approach handling the full transactional Web service design process is available. Some approaches have given formal semantics for the fault and compensation handlers and encoded them in other formal techniques like Petri nets [14, 19], pi-calculus [13], ASM [11], StaAC and B [8] or SAL [17]. In these approaches, the designer describes by himself/herself the parts that are handled by the fault and compensation handlers, and they check properties related to the behaviour like deadlock freeness. There is no systematic formal modelling approach for handling transactions. Moreover, there is no way to detect the transactional part to be isolated in order to guarantee a correct behaviour of the transactional Web service. Furthermore, existing approaches do not provide the possibility to check if the consistency of the execution context is guaranteed. This is due to the abstraction of data in the model checking techniques applied to Web service validation for reducing the state space exploration. As a consequence, these works don't describe nor check properties related to the consistency of data produced by the BPEL processes.

Our work is proof oriented; it translates the BPEL language and its constructs into an Event-B model. We encode manipulated data and transactional behaviour, check traditional properties like deadlock freeness and transactional properties [5, 6] and offer to the developer assistance to improve his/her BPEL design by isolating transactional parts to ensure a correct process behaviour. Moreover, it is tool supported.

Notice that this work applies for all transactional-based process compositions.

9 Conclusion

In this paper, we propose an extension of the BPEL Event-B semantics proposed in [5] and [6] by covering the *scope*, the fault and the compensation handlers. We have also sketched a methodology showing how the obtained Event-B model can be used to handle a transactional behaviour in Web services. Transactional services that access and manage critical resources are isolated in a *scope* elements with compensation and fault handlers. When modelling fault and compensation

handlers by a set of events, it becomes possible to model and check the properties related to transactional Web services. Moreover, the obtained results are not specific to Web service compositions. These results can be reused for the definition of transactional service compositions that occur for areas like telecommunication and network, manufacturing or scheduling. Indeed, the fact that services are Web based is not specific to the proposed approach. BPEL is used as a language for service composition description whatever is the nature or the application domain of the manipulated services.

This work opens several perspectives. One of them is related to the explicit semantics carried by the services. For example, composing in sequence a service that produces distances expressed in centimetres with another one consuming distances expressed in inches should not be a valid composition. Up to now, our approach handles implicit semantics only; it does not handle such a composition. Formal knowledge models carried out by ontologies expressed besides the Event-B models should be investigated.

References

1. Abrial, J.R.: *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, New York (1996)
2. Abrial, J.R.: *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge (2010)
3. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: application to Event-B. *Fundam. Inform.* **77**, 1–28 (2007)
4. Ait-Ameur, Y., Baron, M., Kamel, N., Mota, J.M.: Encoding a process algebra using the Event B method. *Int. J. Softw. Tools Technol. Transfer* **11**(3), 239–253 (2009)
5. Ait-Sadoune, I., Ait-Ameur, Y.: A proof based approach for modelling and verifying web services compositions. In: *14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 1–10. IEEE Computer Society, Potsdam (2009)
6. Ait-Sadoune, I., Ait-Ameur, Y.: Stepwise design of BPEL web services compositions, an Event B refinement based approach. In: *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 51–68, Montreal (2010)
7. Borger, E., Thalheim, B.: Modeling workflows, interaction patterns, web services and business processes: the ASM-based approach. In: *Abstract State Machines, B and Z (ABZ 2008)*. Lecture Notes in Computer Science, vol. 5238. Springer, Heidelberg (2008)
8. Butler, M., Ferreira, C., Ng, M.Y.: Precise modelling of compensating business transactions and its application to BPEL. *J. Univers. Comput. Sci.* **11**(5), 712–743 (2005)
9. Dijkstra, E.W.: *A Discipline of Programming*, 1st edn. Prentice Hall PTR, Upper Saddle River (1977)
10. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: the negative Control Flow. In: *12th International Workshop on Abstract State Machines*, pp. 131–151 (2005)
11. Farahbod, R., Glasser, U., Vajihollahi, M.: An abstract machine architecture for web service based business process management. In: *Business Process Management Workshops*. Lecture Notes in Computer Science, vol. 3812, pp. 144–157. Springer, Heidelberg (2005)
12. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based verification of web service compositions. In: *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pp. 152–163 (2003)

13. Guidi, C., Lucchi, R., Mazzara, M.: A formal framework for web services coordination. *Electron. Notes Theor. Comput. Sci.* **180**, 55–70 (2007)
14. He, Y., Zhao, L., Wu, Z., Li, F.: Formal modeling of transaction behavior in WS-BPEL. In: *International Conference on Computer Science and Software Engineering (CSSE 2008)* (2008)
15. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to petri nets. In: *Springer-Verlag* (ed.) *3rd International Conference on Business Process Management. Lecture Notes in Computer Science*, vol. 2649. Springer, Heidelberg (2005)
16. Hoare, C.A.R.: An axiomatic basis for computer programming. *Commun. ACM* **12**, 576–580 (1969)
17. Kovacs, M., Varro, D., Gonczy, L.: Formal analysis of BPEL workflows with compensation by model checking. *Int. J. Comput. Syst. Sci. Eng.* **23**(5), 35–49 (2008)
18. Leuschel, M., Butler, M.: ProB: a model checker for B. In: *Formal Methods, International Symposium of Formal Methods Europe (FME'03). Lecture Notes in Computer Science*, vol. 2805, pp. 855–874. Springer, Heidelberg (2003)
19. Lohmann, N.: A feature-complete petri net semantics for WS-BPEL 2.0. In: *Web Services and Formal Methods International Workshop WSFM 2007* (2007)
20. Marconi, A., Pistore, M.: Synthesis and composition of web services. In: *Formal Methods for Web Services - 9th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Web Services. Lecture Notes in Computer Science*, vol. 5569. Springer, Heidelberg (2009)
21. Nakajima, S.: Model-checking behavioral specification of BPEL applications. *Electron. Notes Theor. Comput. Sci.* **151**, 89–105 (2006)
22. OASIS: Web Services Business Process Execution Language Version 2.0. <http://bpel.xml.org/> (April 2007)
23. OMG: Business Process Model and Notation (BPMN) Version 2.0. <http://www.omg.org/spec/BPMN/2.0> (June 2010)
24. Rodin: User Manual of the RODIN Platform. <http://deploy-eprints.ecs.soton.ac.uk/11/1/manual-2.3.pdf> (October 2007)
25. Salaun, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra. In: *IEEE International Conference on Web Services (ICWS'04)*, pp. 43–51 (2004)
26. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Formal methods for service composition. *Ann. Math. Comput. Teleinformatics* **1**(5), 1–14 (2007)
27. van der Aalst, W.M., Mooil, A.J., Stahl, C., Wolf, K.: Service interaction: patterns, formalization, and analysis. In: *Formal Methods for Web Services - 9th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Web Services. Lecture Notes in Computer Science*, vol. 5569. Springer, Heidelberg (2009)
28. W3C: Web Service Definition Language (WSDL 1.1). <http://www.w3.org/TR/wsdl> (February 2004)
29. W3C: OWL-S: Semantic Markup for Web Services. <http://www.w3.org/Submission/OWL-S/> (November 2004)
30. W3C: Web Services Choreography Description Language Version 1.0. <http://www.w3.org/TR/ws-cdl-10/> (November 2005)
31. WMC-WS: Process Definition Interface - XML Process Definition Language. <http://www.wfmc.org/xpdl.html> (October 2008)

Towards a Model of Services Based on Cocreation, Abstraction and Rights Distribution

Maria Bergholtz, Birger Andersson, and Paul Johannesson

Abstract The term service is today defined and used in a multitude of ways, which are often ambiguous and contradictory. The absence of a commonly agreed-upon definition of the term makes it difficult to distinguish, describe and classify services. In order to address these issues, this chapter proposes a model of services that helps in analysing the concept. The model encompasses three perspectives: service as a means for cocreation of value, service as a means for abstraction and service as a means for distributing rights. The model does not suggest a definition of the term service but shows how the service concept can be analysed using a number of related concepts, like service resource, service process and service offering. The model has its theoretical foundation in the Resource-Event-Agent (REA) ontology and Hohfeld's classification of rights.

1 Introduction

The increasing interest in services has created a multitude of alternative views and definitions, often conflicting, of the service concept. What constitutes a service is still a matter of debate, in industry as well as in various research communities. The lack of a common view of the service concept makes it difficult to reason about, describe and classify services in a uniform way. One approach to structuring services is to divide them into business services and software services. OASIS [16] and Preist [18] focus on a business service perspective, while [20] has a software service perspective. New methods have also been proposed to structure systems by means of service architectures [3, 7, 17, 23]. For example, in the view of Papazoglou and Van den Heuvel [17] (software), service design and development is about identifying the right services, organising them in a manageable hierarchy of composite services and choreographing them together for supporting a business process. However,

M. Bergholtz (✉) • B. Andersson • P. Johannesson
Department of Computer and Systems Sciences, Stockholm University, Isafjordsgatan 32, 164 40
Kista, Sweden
e-mail: maria@dsv.su.se; ba@dsv.su.se; pajo@dsv.su.se

identifying the right services, or classifying them, is a difficult task due to the aforementioned lack of a common view of the service concept.

One attempt to defining services has focused on identifying properties (such as intangibility, inseparability, heterogeneity and perishability [22]) that distinguish them from other kinds of resources. However, Edvardsson et al. [6], Goldkuhl and Röstlinger [11], Sampson and Froehle [19], Ferrario et al. [8] and others have argued that this approach is problematic in that the suggested properties are neither necessary nor sufficient in terms of defining a service. For example, not only services are intangible but also other kinds of resources, such as information and IPRs. Heterogeneity can be observed also in the production of certain goods and information, such as handicraft objects and newspaper articles.

Instead of attempting to identify services by internal properties that uniquely distinguish them from other kinds of resources, we suggest to identify services through the roles they play for the use and offering of resources [6]. Thus, the focus is shifted from the internal characteristics of resources to their context of use and exchange. This view is shared by the Unified Services Theory [19], which also bases its definition of services on the use and exchange of resources; here, service processes are processes where customers always provide significant input resources, as opposed to non-service processes where customers only select what output resources to buy and pay for.

Services may also be understood as a means for abstraction. A common view found in [14, 16, 18, 20] is services as an abstraction of activities that once started will achieve some user goal, usually defined as a change of state in (user) resources. Ferrario et al. [8], however, argue that a service cannot be defined only in terms of resource-changing activities. An example is a snow removal service, which only guarantees to keep some streets from snow. If it does not snow, no service will be delivered, yet the streets are indeed free from snow. The paradox is that sometimes the terms of a service can be honoured even if no service is actually delivered, i.e. no activity has been executed. It can be observed that many categories of services are analogous to this example, for instance, health care and fire brigade services.

An often mentioned advantage of services is that the management (infrastructure, maintenance, technology, etc.) of resources is moved from customer to provider [4, 5, 22]. This is a consequence of the principle that service provision does not entail ownership transfer [22]. The concept of service can in fact be used as a means for providing restricted resource access without ownership transfer [4, 5]. Resource access is closely related to the various rights an agent is given with respect to a resource. Services may not only provide access to resources but also distribute different types of rights to the resource.

The diversity of service views and definitions and the fact that these views are often conflicting suggest that a multi-perspective approach is required. We will follow this line of reasoning and introduce a number of service perspectives rather than propose a single service definition. We identify three main service perspectives from the literature introduced above: service as a means for cocreation of value [14, 19], service as a means for abstraction [14, 16, 18, 20] and service as a means for distributing rights [4, 5, 22]. The purpose of the chapter is to propose a conceptual

model of services based on these three perspectives. The model has its theoretical foundation in the REA ontology [15] and Hohfeld’s classification of rights [12]. REA is used because it is a well-established ontology of business collaboration with the basic view that resources are exchanged between agents according to agreements. Hohfeld’s classification of rights is used as means for analysing what kinds of rights are transferred in exchanges of services and other kinds of resources. The work reported here builds on the work of [4] and [5], both of which are based on a multi-perspective view of analysing services. The main differences with respect to [4, 5] are (1) a new foundation for the model based on distinguishing between service deliveries and deliveries of goods and other types of resources, (2) a new analysis of the fulfilment of service deliveries versus deliveries of goods and (3) the alignment of the added model concepts with the core REA model.

The remainder of this chapter is structured as follows: In Sect. 2, we briefly outline the main points of the REA ontology and Hohfeld’s classification of rights. In Sect. 3, we introduce the three perspectives of services and elaborate them together with their corresponding conceptual models in Sects. 4, 5 and 6. In Sect. 7, we discuss related work and conclude the chapter.

2 The REA Ontology and Hohfeld’s Classification of Rights

The REA (Resource-Event-Agent) ontology was originally formulated in [15] and developed further in a series of papers, e.g. [10, 13]. The REA ontology is based on the core concepts of resources, events and agents, which are described in Sects. 2.1 through 2.5. Figure 1 shows the ontology (including adaptations and extensions discussed) as a UML (Unified Modelling Language) class schema.

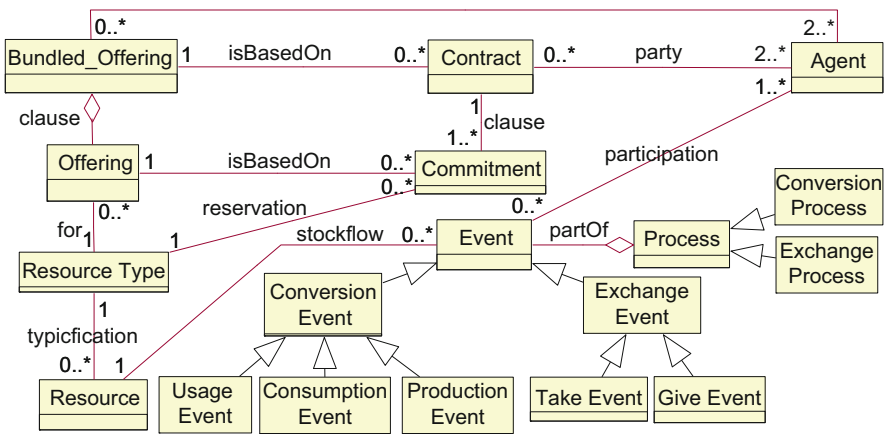


Fig. 1 REA ontology (adopted and extended from <http://reatechnology.com/what-is-rea.html>)

2.1 Resources

A resource is something that is of value for at least one agent, e.g. a car, cloud storage or a stream of music. For the purpose of analysing services, three categorisations of resources are introduced. One first distinction is the one between economic resources and noneconomic ones, where the former are entities that can be controlled by an agent and traded between agents. Resources can also be categorised into assets and consumables depending on the role they play when used in an activity [9]. A consumable is consumed when used in an activity, e.g. in surgery, blood plasma is consumed. In contrast, an asset can be reused, e.g. a nurse can participate in many surgeries. Finally, based on the degree to which a resource is tied to an agent, resources can be classified into independent resources, internal resources and shared resources. These three categories are relevant for analysing services, as services are often used as instruments for distributing rights to resources that are tightly tied to one actor.

An *independent resource* is a resource that can exist independently of any agent. In other words, an independent resource can exist even if it is unrelated to any agent. Typical examples of independent resources are physical objects, land and information.

An *internal resource* is a resource whose existence depends on one single agent. If the agent ceases to exist, so does the internal resource. Examples of internal resources are capabilities, skills, knowledge, memories and experiences. These kinds of resources are dependent on individuals, but in a transferred sense, they can also be dependent on organisations. Furthermore, for organisations, even processes, practices and procedures can be seen as internal resources. A characteristic of an internal resource is that it is not an economic resource, i.e. it is not tradable.

A *shared resource* is a resource whose existence depends on at least two agents. The most common shared resources are relationships and rights. Some relationships are narrow in scope and primarily govern and regulate activities for some particular resource(s), e.g. ownership of goods or a sales order. Other relationships have a wider scope, e.g. a marriage or an employment relationship that includes a large number of rights. Rights will be further discussed in Sect. 2.4.

2.2 Conversion Processes

Resources can be transformed, i.e. they can be produced, modified, used or consumed. Resources are transformed in so-called conversion processes consisting of conversion events. A *conversion event* is a transformation of a single resource. If the conversion event creates a new resource or increases the value of an existing resource, the conversion event is a *production event*. If the conversion event consumes a resource or decreases the value of a resource without consuming it, the conversion event is a *consumption event* or a *usage event*, respectively. Usage

events are using resources that may be reused in several conversion events (similar to the concept of assets [9]), while consumption events use up resources (similar to the concept of consumables [9]). Examples of conversion events are the production of a car, the repair of a car and the consumption of a litre of gasoline.

A *conversion process* is a set of conversion events including at least one production event and at least one consumption or usage event. The latter requirement expresses a duality relationship between production and consumption/usage events, stating that in order to produce or improve some resource, other resources have to be used or consumed in the process. For example, in order to produce a car, a number of other resources have to be used, such as steel, knowledge and labour.

2.3 Exchange Processes

Resources can also be exchanged between agents, which occurs in exchange processes consisting of exchange events. An *exchange event* is the transfer of rights on some resource to or from an agent. If the exchange event means that the agent receives rights on a resource, the event is a *take event*. If the exchange event means that the agent gives up rights on a resource, the event is a *give event*.

An *exchange process* is a set of exchange events including at least one give event and one take event. Similarly to conversion processes, this requirement expresses a duality relationship between take and give events—in order to receive a resource, an agent has to give up some other resources. For example, in a goods purchase (an exchange process), a buying agent has to provide money to receive some goods. Two exchange events take place in this process: one where the amount of money is decreased (a give event) and another where the amount of goods is increased (a take event). The same resource can participate in different types of events. For example, a machine is first acquired (take event), then employed in production (usage event) and finally sold (give event).

2.4 Hohfeld's Classification of Rights

In the sections above, the notion of rights has been used in an informal way. As a more precise understanding of rights will be required for characterising different kinds of resources and exchanges, a rights classification based on the work of W.N. Hohfeld [12] is introduced—Hohfeld identified four broad categories of rights: claims, privileges, powers and immunities.

- One agent has a *claim* on another agent if the second agent is required to act in a certain way for the benefit of the first agent, typically by carrying out some action. Conversely, the second agent is said to have a duty, or an obligation, to

the first agent. An example is a person who has a claim on another person to pay an amount of money, implying that the other person has a duty to pay the amount.

- An agent has a *privilege* on an action if he/she is free to carry out that action without any interference from the environment in which the action is to be carried out. Environments here meant social structures such as states, organisations or even families. Some examples of privileges are free speech and the permission for a person owning some property to use it in various ways.
- A *power* is the ability of an agent to create or modify a relationship. An example is that a person owning a piece of land has the power to sell it to someone else, thereby creating a new ownership relationship for the land.
- An *immunity* refers to the restriction of power of one agent in terms of creating formal relationships on behalf of another agent. For example, native people may hold immunity towards state legislation concerning their property rights, meaning that the state does not have the power to enforce laws that modify existing property rights. (Immunities will not be used in this chapter.)

Most relationships are governed by a combination of several of these kinds of rights. For example, owning a car means to have privileges on using it and also the power to lend or sell it, i.e. creating new ownerships involving other agents.

2.5 Offerings, Commitments and Contracts

Exchange processes can be governed by agreements that specify when and how resources are to be exchanged. The two most important types of agreements are offerings and contracts consisting of commitments. A *commitment* on a resource type is a duty for an agent to carry out a conversion or exchange event for an instance of that resource type. For example, an agent may have a duty towards another agent to transfer the ownership (a give event) of a car (instance of a car type) to that agent. A *contract* is a collection of commitments and possibly additional rules governing their interrelationships.

An *offering* for a resource type is a conditional obligation for one agent to some community of agents to enter into a commitment for that resource type. For example, an agent may provide an offering for a certain car model, meaning that he/she is prepared to sell cars of that model, i.e. enter into commitments for the car model. An offering is similar to a commitment but differs by not being binding until another agent has accepted it. Thus, when an offering is accepted, it will result in a commitment. A set of offerings can be collected into a *bundled offering*, analogously to a contract.

Figure 1 summarises the notions introduced so far in the form of a UML class diagram. In the following sections, these notions will be further analysed and specialised in order to clarify the different perspectives on services. Almost all of the concepts in the conceptual model presented here may exist on both a knowledge level and an operational level. According to [9], the operational level

models concrete, tangible individuals in a domain. In contrast, the knowledge level models information structures that characterise categories of individuals on the operational level. The diagrams of Figs. 1, 2, 3, 4, 5, and 6 hence distinguish between concepts such as resource types (categories of resources such as car model, agent type, real estate) and resource (specific and often tangible concepts like a specific car or a concrete piece of land), event types and events and so forth for every concept in the model. In order to make the diagrams less cluttered, both knowledge and operational level concepts are included only when they are required to illustrate a key issue in the model.

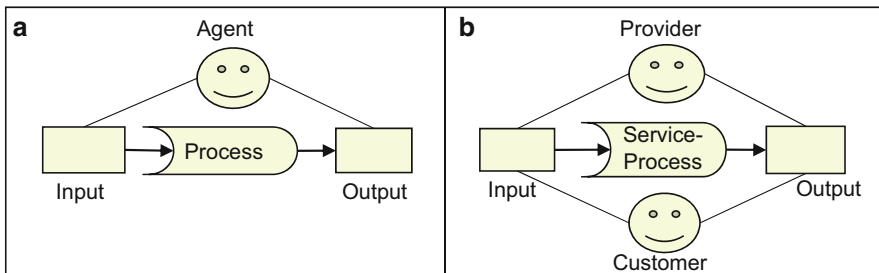


Fig. 2 Single agent process versus service process

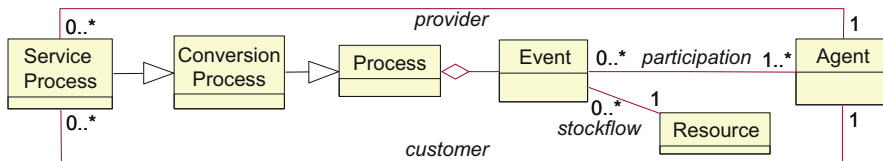


Fig. 3 REA ontology from Fig. 1 expanded with service process to highlight cocreation of value between provider and customer

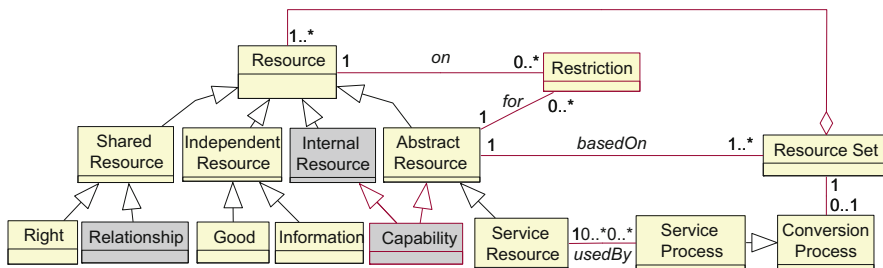


Fig. 4 Service as an abstraction mechanism (noneconomic resources in grey)

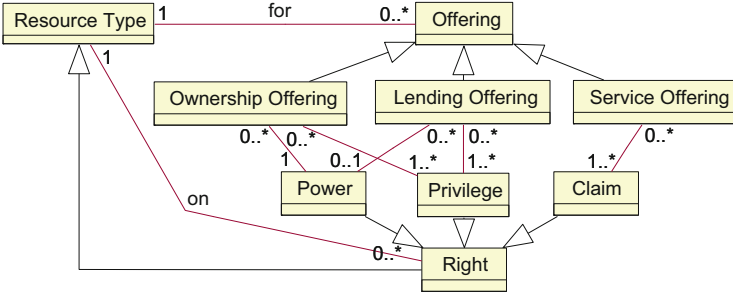


Fig. 5 Service as a means for distributing rights to resources

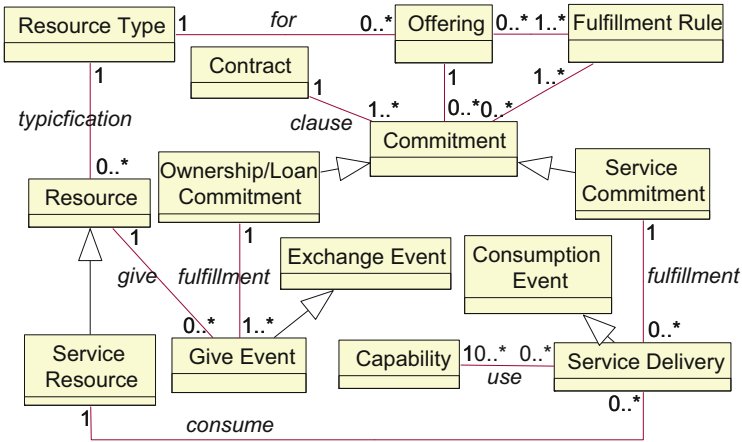


Fig. 6 Fulfilling service commitments (agent relationships removed for simplicity)

3 Service Perspectives

In the following sections, a conceptual model for services will be introduced. The model does not propose a single service definition but instead suggests a number of service perspectives based on the ways resources can be used and exchanged. This approach is reflected in the model, which does not include the term “service” but instead a family of related terms, including “service resource”, “service process”, “service offering” and “capability”. Three main perspectives on services are identified: service as a means for cocreation of value [14, 19], service as means for abstraction [14, 16, 18, 20] and service as a means for distribution of rights [4, 5, 22].

- Service as a means for *cocreation of value*. For most kinds of goods and information, customers are not involved in their production. Instead, goods and information are produced internally at a supplier who later on sells them to a

customer who uses them without the involvement of the supplier. In contrast, services are created and used in an interaction between supplier and customer.

- Service as a means for *abstraction*. Services can provide an abstraction mechanism where resources are specified through their function and not their construction. In other words, a resource is defined in terms of the effects it has in a process, not in terms of its properties or constituents. For example, a hairdressing service can be defined in terms of the effects it has on someone's hair, not in terms of the resources being used in its execution.
- Service as a means for *rights distribution*. An agent can transfer access and rights to some of his/her resources to another agent by transferring the ownership of them. However, such an ownership transfer may in some situations be undesirable or even legally impossible. Thus, there is a need for a way of offering rights on resources without transferring ownership. Services provide a mechanism for this purpose: instead of transferring ownership, a customer is given only limited rights on the resources. For example, instead of selling people, labour services are sold, and instead of selling cars, car rental services are provided.

The model, based on these three perspectives, will be presented in a series of diagrams, all of which have the REA ontology as their point of departure. Figures 2 and 3 show services as cocreation of value, while Figs. 4 and 5 show services as abstraction mechanisms and instruments for rights distribution. The last one, Fig. 6, shows how distribution of rights to resources can be fulfilled.

4 Service as a Means for Cocreation

For a typical goods-producing company, its interactions with customers can be quite limited. Without any involvement from the customers, the company procures raw materials and assets from suppliers, uses these resources to produce goods to be sold and distributes the goods to retailers and other outlets. The only role of the customer is to select which goods to purchase and to pay for them. Thus, the company carries out a conversion process in isolation transforming input resources to output resources; see Fig. 2a.

In contrast to a goods-producing company, a service provider always has to work closely with its customers. A service can never be carried out by a provider in isolation, as it always requires a customer to take part in the process, at least in the sense of providing input resources. In such a service process, the provider and the customer together cocreate value, as both of them provide resources to be used or consumed in the process. For example, in a photo-sharing service, the service provider will supply hardware and software, while the customer will provide photos and labour. Together, they engage in a process that results in value for the customer, shareable photo albums. This process can be compared to that of a hardware supplier, who produces computers in isolation from the customer, who will later on buy the finished product and use it without any interaction with the

supplier. Pictorially, a service process can be viewed as in Fig. 2b, which shows how both a service provider and a customer jointly contribute to the service process that produces an output for the benefit of the customer.

In order to make the concept of service as cocreation more precise, it is useful to distinguish between service as a process and service as a resource. The word “service” is sometimes used to denote a process, e.g. in the sentence “Today, our company carried out 25 car repair services”. In other cases, “service” is used to denote a resource, e.g. “Our company offers car repair services for the fixed price of 200 euros”.

A *service process* (see Fig. 3) is a conversion process that uses or consumes resources from two agents, called provider and customer, and produces resources that are under the control of the customer, i.e. the customer has rights on these resources. The provider in the service process has to actively participate in the process, while the customer may be passive (apart from providing input resources). For example, a customer driving a borrowed car does not constitute a service process, while a customer being driven by the provider does so. Thus, a service process differs from other processes in three respects. First, some of the input resources are under the control of one agent, the provider, while the output resources are under the control of another agent, the customer. This means that the provider uses or consumes his/her resources in the service process for the benefit of another agent. Secondly, not only the provider but also the customer provides resources as input to the service process. Thirdly, the provider actively takes part in the service process.

5 Service as a Means for Abstraction

When offering resources, it may seem preferable to provide as much information as possible about them. However, being able to specify resources in an abstract way often provides key advantages. It becomes easier for a provider to describe the benefits of an offering when he/she can focus on the effects of using the resource offered and abstract away from its accidental features. The provider can address the needs and wants of the customer and clarify how these are fulfilled by his/her offering without going into detail about its constituents. Furthermore, the provider does not have to commit to any specific way of delivering his/her offering; instead, he/she can choose to allocate the resources needed in a flexible and dynamic way. In order to manage these kinds of offerings, the notion of abstract resource is useful. An *abstract resource* is a resource that is defined solely in terms of its effects in a conversion process. For example, a laundry service is defined in terms of the effects it has on clothes—making them clean. When an abstract resource is used, there must exist a number of underlying resources that realise it. A set of such resources is modelled by the class *resource set* in Fig. 4. For instance, a laundry service may be based on different resource sets: washing machines, synthetic detergent or water tank, soap and labour.

A *service resource* is an abstract resource that is defined only through its use and effects in a service process, i.e. what changes it can bring to other resources when consumed in such a process. For example, a haircut service is defined through the effects it has on the hairstyle of a person. It is not defined by means of the concrete resources used when cutting the hair, such as labour, scissors and shampoo. Rather, the concrete resources to be used are left unspecified and can change over time. On one day the hairdresser may use scissors and shampoo and on another day an electric machine and soap, but in both cases, he/she provides a haircut service. Thus, the same service resource can be based on different resource sets, but when it is consumed, exactly one of these resource sets will be used.

Although the possibility to specify resources in an abstract manner is a key advantage of using the notion of service resources, there are cases where it is preferable to be more concrete. In particular, it may be desirable to put constraints on the resource sets on which a service resource can be based. For example, a hairdresser may offer a “hair dyeing” service and declare that it is based exclusively on colouring products with environmentally friendly ingredients. In this case, the service resource would be defined not only through its effects but also through constraints on the resource sets on which it is based. In Fig. 4, the class *restriction* is used to represent such constraints.

While the notion of service resources primarily is useful for providing interfaces between agents in the context of resource exchanges, the related notion of capability can help to structure an organisation internally. A *capability* is an internal resource that is defined through the conversion processes in which it can be used. Similarly to a service resource, a capability is abstract in the sense that it is not defined in terms of its properties and components, but by the effects it can produce. In contrast to a service resource, a capability can be used in any process, not only in service processes. Thus, a capability of an agent can be used to produce something that is under the control of that agent. Furthermore, a capability is not an economic resource, i.e. it cannot be traded. Instead, a capability is internal to an agent, meaning that it is dependent on some agent possessing the capability and can be used only when that agent is present. Some examples of capabilities are the ability to provide Internet access, to offer high school teaching and to support marketing campaigns. As in these examples, capabilities are often broadly and vaguely delimited, thereby specifying in general terms what an agent is able to accomplish. Service resources, on the other hand, are typically more precisely delimited as they are to be traded. Therefore, service resources are often used to externalise capabilities by exposing some parts of them.

6 Service as a Means for Distributing Rights

When satisfying a need, an agent can often choose between using a service or some other kinds of resource, like goods or information. Using a service instead of another kind of resource provides several benefits, as the service consumer does not own the

service and therefore does not have to take on typical ownership responsibilities, like infrastructure management, integration and maintenance. Instead, he/she can focus on how to make use of the service for satisfying his/her needs. For example, a person can satisfy his/her transportation needs either by buying and driving a car or by using a taxi service. In the former case, he/she will own the car required for the transportation, meaning that he/she will be responsible for cleaning it, repairing it, getting the right insurances and many other infrastructure and maintenance tasks. When using a taxi service, on the other hand, he/she does not have to care about any of these responsibilities but can focus solely on how to use the taxi to best satisfy his/her transportation needs. Thus, services provide a convenient way of offering and accessing resources by allowing agents to use them without owning them. In other words, the rights on the resources underlying the service are distributed between the provider and customer in such a way that the customer will have convenient access to them without the hurdle of maintaining them.

Figure 5 depicts three different ways for an agent to make its resources available to other agents through offerings, each of them distributing rights in different ways:

- An agent may offer to sell a resource to another agent, i.e. to transfer the ownership of the resource to the other agent, as modelled by *ownership offering*. A transfer of ownership means that all the rights on the resource are transferred from seller to buyer, in Fig. 5, modelled by the class *right*. The rights transferred include powers and privileges according to Hohfeld's classification of rights in Sect. 2.4. As an example, an agent offering to sell a book to a customer means that the agent is offering the customer privileges to use the book as well as the power to transfer the ownership of the book to other agents.
- An agent may offer to lend a resource or provide access to it in a *lending offering*. This means to offer an agent to get certain privileges on the resource for a period of time but without getting any ownership, i.e. the borrower is not granted the power to transfer the ownership of the resource. Optionally, the borrower may get some other powers, such as lending the resource to a third agent.
- An agent may make a *service offering* to a potential customer, which is the way in which the least amount of rights is transferred from the provider to the customer. A service offering means that the provider offers to use some of his/her service resources in a service process that will benefit the customer. In this case, the provider can be seen as standing between the customer and the concrete resources to be used in the service process. Effectively, the provider restricts access to these resources. In particular, the customer is not offered any powers or privileges on any concrete resources. Instead, he/she is offered a claim on the provider to contribute to a certain service process.

In the next section, we will analyse under which types of conditions the rights of an offering are actually transferred to fulfil what the providing agent is offering the customer.

6.1 Fulfilling Commitments

When offerings are accepted, they will result in commitments and contracts; see Fig. 6. Service offerings will result in *service commitments*, while ownership or loan offerings result in *ownership/loan commitments*. When commitments have been established, the providing agent is obliged to fulfil them by carrying out conversion and/or exchange events that consume and/or transfer the (committed) resources to the receiving agent.

Commitments can be fulfilled in different ways depending on the kind of offering they are based on:

- An ownership/loan commitment is fulfilled by an agent carrying out a give event, where the agent gives rights (privileges and/or powers) on the committed resource to another agent.
- A service commitment is fulfilled by an agent carrying out a consumption event, where the committed service resource is consumed in a service process. Such a consumption event is called a *service delivery*. Thus, a service commitment becomes fulfilled through an agent using his/her own resources in order to benefit another agent, i.e. the resources on which the service resource is based.

Summarising, a service commitment is fulfilled by an agent consuming and using his/her own resources, while an ownership/loan commitment is fulfilled by an agent giving away rights.

Every commitment is associated to a *fulfilment rule* that specifies one or several time points before which the commitment has to be fulfilled; see class *fulfilment rule* in Fig. 6. In many cases, this rule is simply an absolute time point, e.g. “15 Feb 2015”. We refer to this type of rule as an *absolute fulfilment rule*, which specifies an absolute time point before which a commitment has to be fulfilled. In more complex cases, a fulfilment rule can include various environmental factors, e.g. “within four hours after more than 5 cm of snow has fallen at any time during 2013” or “when the customer has received a certain diagnosis”. A characteristic of this latter type of rule is that it is conditional. A *conditional fulfilment rule* describes under which conditions the provider has to fulfil a commitment. For example, in offerings of insurances of burglary or health care, the customer does not always get access to the rights offered. To actually receive ownership of money as compensation for lost goods in case of a burglary or to receive a treatment service in case of health care, a burglary has to occur or the customer has to become ill.

A commitment is said to be violated at a certain time point if (one of) the time point(s) given by its fulfilment rule has passed and the commitment is not fulfilled. As a contract contains a number of commitments, a contract is said to be violated at a certain time point if any of its commitments has been violated at that time point.

We are now in a position to resolve the apparent paradox of the snow ploughing case presented in Sect. 1. The key to the solution is to distinguish between service deliveries and service contracts. A service contract can be respected, i.e. not violated, even though none of its commitments is ever fulfilled. This is exactly

what would happen in the case where no snow falls during winter. As there is no snow, no commitment will ever need to be fulfilled, i.e. no service resource will be consumed. Still, the service contract is respected, as no commitment is ever violated. An equivalent example is the service resource health care, where the service contract is respected if either the customer does not fall ill and no service delivery is required or if the customer does fall ill and a service delivery actually occurs. In summary, service contracts containing conditional fulfilment rules may be respected even though no service deliveries ever occur.

7 Concluding Remarks

In this chapter, we have proposed a conceptual model of the notion of service. A main characteristic of the model is that it describes services from three perspectives—service as a means for cocreation of value, for abstraction and for rights distribution. The work was in part motivated by a problem posed in [8]. The issue there was how to view a service where the terms of the service could be honoured even if no service is actually delivered. The apparent paradox was resolved by distinguishing between service contracts and service deliveries. The work is moreover motivated by the assumption that cocreation of value is fundamental for services as argued in [19]; in other words, taking into account only one agent’s perspective at a time is not sufficient when modelling services.

Our three perspectives can be compared to those introduced in [1]. There the chosen perspectives are called “service value”, “service offering” and “service process”. The service value perspective is analogous to our abstraction perspective, where a service is described by the effects it produces, but it also contains elements from our co-production perspective. The service offering perspective is related to our view of services as a means for restricted access to resources. The service process perspective describes how a service offering is put into operation, but in contrast to our proposal, the authors do not investigate realisation issues in detail.

In the context of SOA, OASIS acknowledges that services are not only a technical but also a social concept [16]. It is stated that many, if not most, effects that are desired in the use of SOA-based systems are actually social effects rather than physical ones. When a customer “tells” an airline service that it “confirms” the purchase of the ticket, it is simultaneously a communication and a service action—“two ways of understanding the same event, both actions, one layered on top of the other, but with independent semantics” [16] (p. 32). Compared to our three perspective views, OASIS focuses on abstraction and access restriction (of mainly software services). Lusch [14], on the other hand, emphasises the cocreation of value perspective and argues that it is paramount for a so-called service-dominant logic, which can be contrasted with a goods-dominant logic. Alter [2], in discussing SOA (Service Oriented Architecture) and SOE (Service-Oriented Enterprise), also stresses the cocreation element (as well as provisioning of resources) in a general definition of services: “Services are acts performed for other entities including the provision of resources that other entities will use” [2].

An additional motivation for the work presented here was inspired by a language problem identified by Wittgenstein [21]. He/She contends that a word is defined by its use, that it can be used in different ways and that there is no usage characteristic that is common for all these ways. He/She likens the different uses with a family of meanings of the word. In the context of services, this is particularly problematic since no common agreed-upon definition of the term exists and the natural language terms used are often misleading. Analysing services along the dimensions cocreation, abstraction and restriction mechanisms makes it possible to distinguish between similarly labelled but different concepts. For instance, a “health-care insurance service” is different from a “burglar insurance service” (the latter refers to the dimension of customer participation and hence is not a service process). The analysis also shows that it is not meaningful to classify entire industrial sectors such as entertainment, restaurants, insurance, rental services, etc., as belonging to the service sector. Any industrial sector rather offers service resources as well as other kinds of resources. Our analysis can be used as an instrument to classify what resources and processes in the sectors are service resources and service processes, respectively.

In addition to their theoretical contributions, we believe that the results of the chapter will find applications in structuring service descriptions and developing service classifications. Further research will investigate these issues as well as consolidate the proposed model.

References

1. Akkermans, H., Baida, Z., Gordijn, J., Peiia, N., Altuna, A., Laresgoiti, I.: Value Webs: using ontologies to bundle real-world services. *IEEE Intell. Syst.* **19**(4), 57–66 (2004)
2. Alter, S.: Genuinely service-oriented enterprises: using work system theory to see beyond the promise of efficient software. In: *Proceedings of AMCIS 2012, the Eighteenth Americas Conference on Information Systems*, Seattle, Washington (2012)
3. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. *IBM Syst. J.* **47**(3), 377–396 (2008)
4. Bergholtz, M., Andersson, B., Johannesson, P.: Abstraction, restriction, and co-creation: three perspectives on services. In: Trujillo, J., Dobbie, G., Kangassalo, H., Hartmann, S., Kirchberg, M., Rossi, M., Reinhartz-Berger, I., Zimányi, E., Frasincar, F. (eds.) *Advances in Conceptual Modeling – Applications and Challenges*. Lecture Notes in Computer Science, vol. 6413, pp. 107–116. Springer, Heidelberg (2010)
5. Bergholtz, M., Andersson, B., Johannesson, P.: Towards a model of services based on co-creation, abstraction and restriction. In: *Proceedings of the 30th International Conference on Conceptual Modeling, ER’11*, pp. 107–116. Springer, Heidelberg (2011)
6. Edvardsson, B., Gustafsson, A., Roos, I.: Service portraits in service research: a critical review. *Int. J. Serv. Ind. Manag.* **16**(1), 107–121 (2005)
7. Erl, T.: *Soa: Principles of Service Design*. Prentice Hall Press, Upper Saddle River (2007)
8. Ferrario, R., Guarino, N., Fernández-Barrera, M.: Towards an ontological foundation for services science: the legal perspective. In: Sartor, G., Casanovas, P., Biasiotti, M., Fernández-Barrera, M. (eds.) *Approaches to Legal Ontologies*. Law, Governance and Technology Series, vol. 1, pp. 235–258. Springer, Netherlands (2011)
9. Fowler, M.: *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading (1997)

10. Geerts, G.L., McCarthy, W.E.: An ontological analysis of the economic primitives of the extended-REA enterprise information architecture. *Int. J. Account. Inf. Syst.* **3**(1), 1–16 (2002)
11. Goldkuhl, G., Röstlinger, A.: Beyond goods and services - an elaborate product classification on pragmatic grounds. In: *Proceedings of Quality in Services (QUIS 7)*, Karlstad University (2010)
12. Hohfeld, W.N.: Fundamental legal conceptions as applied in legal reasoning. Reprint from 23 *Yale Law Journal* (1913) (1978)
13. Hruby, P.: *Model-Driven Design of Software Applications with Business Patterns*. Springer, Heidelberg (2006)
14. Lusch, R.F., Vargo, S.L., Wessels, G.: Toward a conceptual foundation for service science: contributions from service-dominant logic. *IBM Syst. J.* **47**(1), 5–14 (2008)
15. McCarthy, W.E.: The REA accounting model: a generalized framework for accounting systems in a shared data environment. *Account. Rev.* **57**(3), 554–578 (1982)
16. OASIS: Reference Model for Service Oriented Architecture 1.0. <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf> (2006)
17. Papazoglou, M.P., Van Den Heuvel, W.J.: Service-oriented design and development methodology. *Int. J. Web Eng. Technol.* **2**(4), 412–442 (2006)
18. Preist, C.: A conceptual architecture for semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) *The Semantic Web – ISWC 2004. Lecture Notes in Computer Science*, vol. 3298, pp. 395–409. Springer, Heidelberg (2004)
19. Sampson, S.E., Froehle, C.M.: Foundations and implications of a proposed unified services theory. *Prod. Oper. Manag.* **15**(2), 329–343 (2006)
20. W3C: Web Services Architecture W3C Working Group. <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/> (2004)
21. Wittgenstein, L.: *The Blue and Brown Book 1933–1934*. Harper & Row, New York (1980). Available online at http://www.geocities.jp/mickindex/wittgenstein/witt_blue_en.html (1933)
22. Zeithaml, V.A., Parasuraman, A., Berry, L.L.: Problems and strategies in services marketing. *J. Mark.* **49**(2), 33–46 (1985)
23. Zimmermann, O., Krogdahl, P., Gee, C.: *Elements of Service-Oriented Analysis and Design*. <http://www.ibm.com/developerworks/webservices/library/ws-soad1/> (2004)

Integrating a Model-Driven Approach and Formal Verification for the Development of Secure Service Applications

Marian Borek, Kuzman Katkalov, Nina Moebius, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel

Abstract We present SecureMDD, a development method for secure service applications that integrates a model-driven approach with formal specification techniques using abstract state machines (ASMs), refinement to code and verification with the interactive theorem prover KIV. A larger case study is used to highlight various aspects of the method with a focus on services and their formal verification.

1 Introduction

Distributed security-critical applications with different communicating components like (Web) services, computers, terminals, or smart cards rely on cryptographic protocols. However, the development of such protocols is notoriously difficult and error prone [2, 52]. This is true even for short protocols with only few communication steps [37]. The reason is the presence of a human attacker who actively tries to break security by eavesdropping and modifying the communication between two components. Often, flaws in an application or in the underlying protocols are detected only after years of usage, e.g., in the Europay–MasterCard–Visa (EMV) protocol [48] used in millions of debit and credit cards or in the Transport Layer Security (TLS) protocol [55].

To be able to develop secure applications based on cryptographic protocols, it is essential to integrate formal verification into the development process. Moreover, the security aspects of the application under development have to be considered in all phases of the development process. SecureMDD is a model-driven development method that realizes both aspects and is tailored to develop security-critical smart card and service applications. Moreover, executable code that is correct and secure with respect to the formal model is generated automatically. This eliminates the problem that buggy implementations of secure protocols often render the application insecure.

M. Borek • K. Katkalov • N. Moebius • W. Reif • G. Schellhorn • K. Stenzel (✉)
Institute for Software and Systems Engineering, Augsburg University, 86135 Augsburg, Germany
e-mail: stenzel@informatik.uni-augsburg.de

This chapter focuses on (Web) services in SecureMDD. Service-oriented architectures (SOA) are a common way to develop business or e-government applications. Functionalities are deployed as exchangeable services that can be reused and orchestrated to complex systems. Many languages, standards (e.g., Web Services Business Process Execution Language (WS-BPEL), Service-oriented architecture Modeling Language (SoaML), Web Services Description Language (WSDL), Business Process Model and Notation (BPMN)), and approaches [4, 24, 38] exist to develop such systems. Standard security aspects are covered by existing standards such as WS-Security [49] and WS-SecurityPolicy [50] and the use of standard security protocols like TLS [20]. However, using such application-independent standards and protocols is not sufficient to guarantee the security of an application.

Our approach to develop security-critical service applications allows to completely model the whole system with Unified Modeling Language (UML). Thus, in contrast to other approaches (e.g., Business Process Execution Language (BPEL)), an implementation of the modeled application can be generated automatically. The manual implementation of method bodies is not necessary. Moreover, from the UML model of the application, we generate a formal specification based on abstract state machines (ASMs, [15]) for interactive verification of application-specific security properties.

SecureMDD started with smart card applications. Services differ a lot from smart cards because of the different communication abilities, the different behaviors, and the different operational areas. In order to integrate services into SecureMDD, it has to be clarified how services, their communication, and their security are modeled, how a code is generated, how their behavior is specified formally, and how their security is verified. Some of these aspects have already been described in [11]. The contribution of this chapter is a detailed explanation of how to formally specify services and a new case study, the first fully verified SecureMDD application using services.

The rest of the chapter is structured as follows: Section 2 gives an overview of the SecureMDD approach and Sect. 3 introduces the case study. Section 4 describes the formal specification and verification of the example. Section 5 explains the code generation as well as its deployment. Section 6 discusses related work, and Sect. 7 concludes this chapter. The full model of the case study, the formal verification, and the generated code can be found on our Web page.¹

2 The SecureMDD Approach

SecureMDD is a model-driven development method to create secure applications based on cryptographic (or more broadly speaking, security) protocols. Figure 1 contains an overview.

¹<http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD/>.

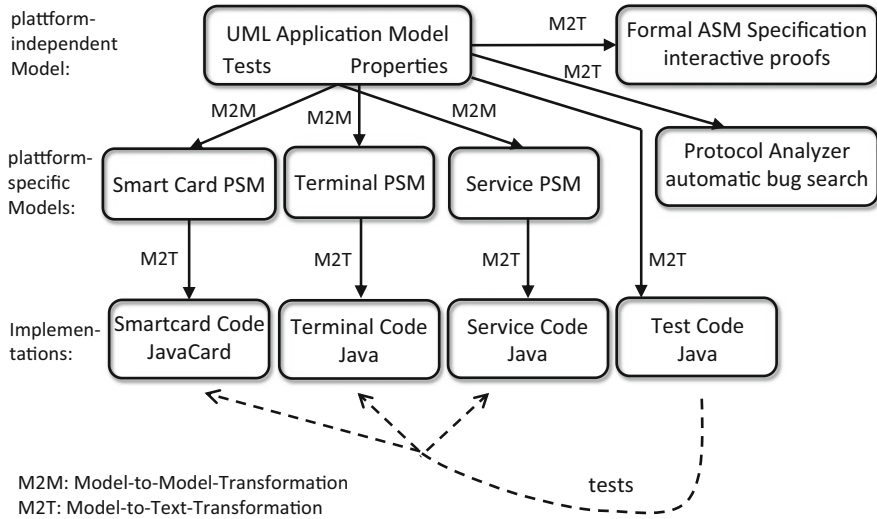


Fig. 1 Overview of the SecureMDD approach

The development of an application starts with the creation of a platform-independent UML model [42, 44]. This is an abstract view of a system, omitting implementation details. To be able to model security-critical applications, UML was tailored to this domain by defining a UML profile. The static part of the application (i.e., components and their attributes, data, etc.) is modeled with a class diagram, and a deployment diagram models the communication structure. To support the modeling of the dynamic part (the communication protocols and behavior of the components) of an application, we defined a domain-specific language called Model Extension Language (MEL) which is used in UML activity diagrams. With this language, it is possible to make assignments to the attributes of component classes, to create objects, or to call predefined cryptographic operations. The platform-independent UML model of an application consists of all information that is needed to generate executable code as well as a formal model of the whole application automatically. An example is presented in the next section.

Additionally, the model contains tests (functional tests as well as attempts to break the security, i.e., attacks) that are modeled with sequence diagrams [34] and application-specific security properties expressed in OCL [13]. Examples for application-specific properties are the following: *An electronic prescription that is stored on an electronic health card cannot be filled twice in a pharmacy, an electronic ticket cannot be forged, and no money is lost in an electronic payment system.* In our opinion, application-specific security properties give better guarantees to the security of an application than standard properties like secrecy, integrity, or authenticity. However, standard properties are often prerequisites for proving application-specific properties and, thus, have to be verified as well. Our approach generates a formal specification based on algebraic specifications and

ASMs for the modeled application. The generated formal model is suitable for the theorem prover KIV [5] and is used for interactive verification of the application-specific security properties [45].

Interactive verification often requires substantial effort, and if an error in the protocol is found, the verification has to start again with the corrected model. In order to detect flaws early and efficiently, the application model can be translated into the input language of the automatic protocol analyzer AVANTSSAR [3]. AVANTSSAR systematically generates all possible traces of the system for a fixed number of components and a fixed number of protocol runs and checks for a violation of security properties. From our experience [12], this can find simple bugs fast, but fails for more intricate errors. One problem is that the search space becomes too big, so that AVANTSSAR does not terminate or runs out of memory. Another problem is that some application-specific properties cannot be expressed in AVANTSSAR, so that only a simplified approximation can be checked. Only the interactive verification fully proves the security of the application.

Runnable code for the application can also be generated from the abstract model. The platform-independent model is transformed by model-to-model transformations into three platform-specific models (PSMs), one for the modeled smart card components, one for the terminals and computers, and one for the modeled services. The PSMs contain the relevant information for one component type and add technical details about the implementation. Using the PSMs as input, executable code of the application is generated automatically using model-to-text transformations. For the smart card components, Java Card [28] code is generated. For the terminals and PCs, we generate Java code. Services are implemented as Java Web Services. Additionally, Java test code is generated from the modeled tests and it is used to test the other codes.

Smart cards are small, secure, tamper-proof devices. They are the basis for many security-relevant applications like debit and credit cards, SIM cards, electronic passports, electronic identity cards, access control, etc. The challenges to generate code for these cards in SecureMDD are explained in [43]. This chapter focuses on services.

The security properties that are proved to hold on the formal model should also hold on the code level. To guarantee this, the generated code has to be a refinement of the generated formal model for every modeled application. This requires the proof that the transformations ensure this refinement relation. However, this is research in progress. The first result is the definition of a calculus for QVT [53] (the language used to implement the model-to-model transformations) in KIV. This calculus can be used to prove the correctness of QVT transformations and of generated Java code [59].

The application model can be created with any UML tool that is compatible with Eclipse. All transformations are realized with the Eclipse modeling framework. QVT [53] is used for the model-to-model transformations and XPand [61] for the model-to-text transformations. All artifacts can be generated by a single click in Eclipse.

3 Case Study: Banking

We present a banking case study that uses smart cards, computers, automated teller machines (ATMs), and different services. As the example is concerned with money, it is security critical. This section shows how such a system is modeled in the SecureMDD approach. The next section describes the formal verification of the example, and then the code generation for services is explained.

The banking case study has two use cases:

- A customer can withdraw money at an ATM. The ATM may belong to the customer's bank or to another bank. The ATM communicates with its own bank, and in the second case, the bank owning the ATM communicates with the customer's bank to authorize the withdrawal.
- A customer can transfer money online to another account that can be located at another bank. The customer uses a PC that communicates with an online banking service which in turn communicates with the customer's bank. In case of an inter-bank transfer, the customer's bank then communicates with the other bank.

The main application-specific security property that the banking systems ensures is the following:

The amount of money in the banking system is constant (if money paid out at ATMs is also counted).

This will be explained in more detail in Sect. 4.6 where the formal specification and verification of this property is described. Next we describe the communication structure of the banking system, then the static view, and finally the protocols.

3.1 Communication Structure

The communication structure and components are defined in a UML deployment diagram (Fig. 2). It consists of nodes, communication paths, and different stereotypes. They are described in turn.

3.1.1 Components

The customer (called `AccountOwner` in Fig. 2) represents a real human being. He/she interacts either with an ATM or a PC that in turn communicates with an `OnlinebankingService`. Both the PC and the ATM need the customer's Debitcard (the protocols will be explained later in this section). In this scenario, we have two types of banks: `AffiliatedBanks` that operate ATMs and `DirectBanks` that only provide online services. For simplicity, we consider only one affiliated and one direct bank in this example, but multiple instances of services

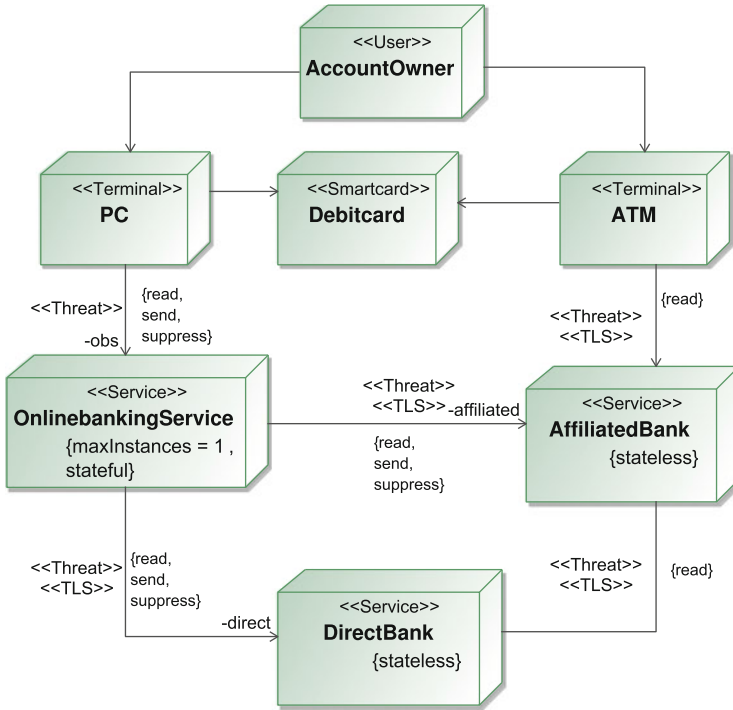


Fig. 2 Deployment diagram for the banking example

are supported in SecureMDD. The online banking service can communicate with both banks, and the two banks can communicate with each other.

The type of each component is indicated by the stereotype in the node. Hence the stereotype <<Service>> shows that OnlinebankingService, DirectBank, and AffiliatedBank are services. A service can be stateful or stateless. OnlinebankingService is stateful. This means it can maintain a session and keeps information like a protocol state and a session key for every invoker. This allows to secure messages between Debitcard and OnlinebankingService with an application-specific cryptographic protocol that uses PC only as an intermediate. The other services (AffiliatedBank and DirectBank) do not need to store session-dependent information, and thus, it is sufficient that they are stateless.

3.1.2 Communication Paths

The communication between components can be unidirectional (indicated by an arrow head) or bidirectional (no arrows). The connection between AccountOwner and ATM is unidirectional. This means that every action is triggered by the user;

only after an ATM has received an instruction from the user will it become active. The same is true between ATM and Debitcard: The ATM will send a message to Debitcard, and afterward the card can answer, but it cannot send messages of its own accord. In contrast, the connection between DirectBank and AffiliatedBank is bidirectional. This means that any of them can start a communication with the other if it has received a message from an ATM or the OnlinebankingService.

3.1.3 Threats

The communication paths also contain «Threat» stereotypes. They describe attacker capabilities for the connection. He/she can be a full Dolev and Yao [21] attacker who is able to *read*, *send*, and *suppress* messages on the fly, but he/she can also have only a subset of these abilities.

The attacker's abilities and connection security influence each other. If a connection has no applied «Threat» stereotype like the connection between AccountOwner and ATM or ATM and Debitcard, then the connection is assumed to be secure (which must be achieved by physical means). A threat between AccountOwner and ATM would mean that the attacker can observe the personal identification number (PIN) the user types (by shoulder surfing or a hidden camera); a threat between Debitcard and PC could be the result of malware on the PC. Both are not considered in the example. The PC communicates with the OnlinebankingService over the Internet. Here, the full Dolev–Yao attacker is assumed (e.g., a malicious employee of the Internet service provider). The connection does not use TLS (see below) because the protocol will realize an end-to-end encryption between Debitcard and OnlinebankingService which makes TLS unnecessary.

3.1.4 Transport Layer Security

If a connection has an applied «Threat» stereotype with any ability, the connection can be secured with TLS (the standard Transport Layer Security protocol [20], indicated by the «TLS» stereotype) with mutual authentication as the default. This means that both communication partners know each other in advance and authenticate each other when the communication starts. An alternative is server-side authentication where only the server authenticates itself. TLS is used for the communication between OnlinebankingService and the banks. The connection has a «Threat» stereotype with the properties *read*, *send*, and *suppress*. That means that the attacker can read, send, and suppress messages on this connection. However, in combination with «TLS», it means that the attacker can only read encrypted messages and that the properties *send* and *suppress* imply that the attacker can only disconnect the connection (see Sect. 4 for more details). The assumption for the connection between DirectBank and AffiliatedBank is that the attacker

can only read messages. Thus, the attacker is not able to affect the connection or the transmitted messages.

3.2 Static View of the Banking System

A UML class diagram specifies all components with their attributes, data types, and messages. Figure 3 shows the relevant part of the class diagram; only the messages are omitted.

3.2.1 Classes

The component classes are annotated with the stereotypes «Service», «Terminal», «Smartcard» as in the deployment diagram (Fig. 2). The two components AffiliatedBank and DirectBank are specializations of the abstract Bank. The other classes are data types that are either stored in attributes or transmitted in messages or both. Messages and their content are also modeled as classes.² Since components send and receive messages, they have a UML

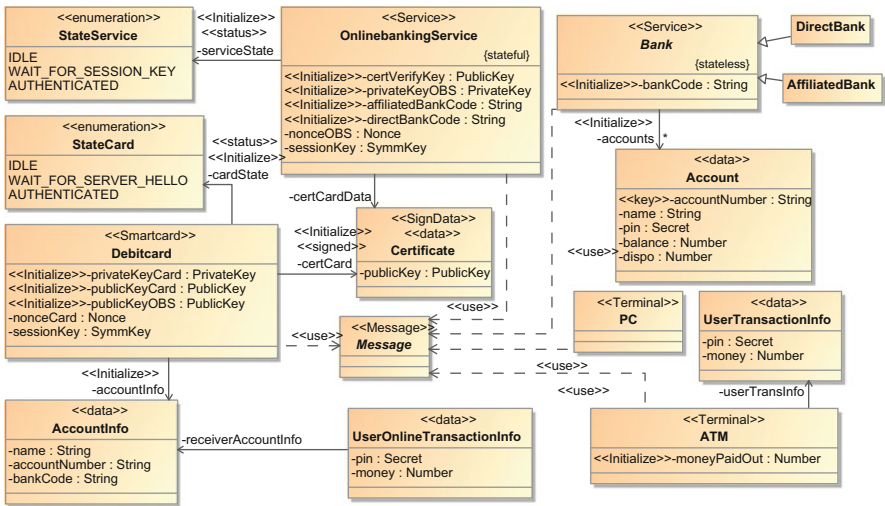


Fig. 3 Class diagram of the banking example

²All 21 messages can be found on our web page <http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD/>. Some of them are used in the activity diagrams in Figs. 4 and 6.

dependency to the abstract message class `Message` from which all messages are derived.

3.2.2 Cryptographic Data Types

The `OnlinebankingService` and the `Debitcard` employ cryptographic keys (`PublicKey`, `PrivateKey`, `SymmKey`), nonces (i.e., random numbers, `Nonce`), and a certificate to establish a secure communication between each other. These data types are predefined in `SecureMDD`. The `Certificate` class is annotated with the stereotype `«SignData»` and the association `certCard` with `«signed»`. This means that the data is not used as clear text but digitally signed, and the signature is used. Other cryptographic data types are `Secret` for information that the attacker should never know and types and operations for encryption and hashing.

3.2.3 Attributes

The `«Initialize»` stereotype indicates that this attribute or association has to be set during the initial deployment of the system. For example, a `Bank` has a code (attribute `BankCode`) and a list of accounts (association `accounts`) that are fixed since we do not consider opening or closing accounts in this example.

The `Debitcard` contains account information of an account owner who should be the owner of the card. They have no keys, states, or nonces because their communication will be secured with TLS. The `PC` has no attributes because it only forwards the messages between `OnlinebankingService` and `Debitcard`. An ATM temporarily stores the PIN and the money that an account owner tries to withdraw in a `UserTransactionInfo` class. Additionally, it also keeps track of how much money it has ever paid out in an attribute `moneyPaidOut`. This information is important to express the main security property mentioned at the beginning (*the amount of money in the banking system is constant*) even though it is not really necessary for the functionality of the system.

3.3 Dynamic View: Protocols and Behavior

The dynamic part of the system is described with activity diagrams. They describe the communication protocols between components and what happens inside a component. The full functionality of the system is defined. All in all, eight activity diagrams are modeled: three communication protocols and five activity diagrams for internal behavior.

3.3.1 Protocol for Withdrawing Money

The activity diagram for withdrawing money from an ATM is shown in Fig. 4. Five components are involved in this scenario: an `AccountOwner`, an `ATM`, a `Debitcard`, an `AffiliatedBank`, and optionally a `DirectBank`. They are modeled as swim lanes. First an `AccountOwner` inserts his/her card into an ATM slot, chooses “withdraw money”, and enters his/her PIN as well as the sum of money to withdraw on the user interface of the ATM (1). The interaction of a real human with the ATM is modeled as a message `UDebit` with argument `userTransactionInfo` (an instance of the class `UserTransactionInfo`) that is sent to the ATM. Message passing is indicated by UML `SendEvent` and `AcceptEvent` nodes. The ATM stores this information in its attribute `userTransInfo`, asks the `Debitcard` for the account information (2), and sends all data to its owner, the `AffiliatedBank` (3). The bank first checks if the account belongs to itself or another bank by comparing the received bank code with its own. This is modeled with a UML decision node and guards. If the account belongs to the bank itself, it calls the `debitFunction` (4) and then, depending on the return value, the money is issued or not. The fork symbol in the node indicates a reference to another activity diagram where the behavior of `debitFunction` is modeled, and the flow `final` node indicates abortion of the protocol. `debitFunction` debits an account if everything is ok (PIN correct and credit line not exceeded). If the account belongs to another bank, the message is forwarded to the `DirectBank` (5) and the amount will be debited there. After that, if the debit action was successful, a message `TerminalPayOut` is sent back (6) and the ATM pays out the money (7).

3.3.2 Details About Protocols in SecureMDD

The actions and guards contain statements of our MEL language. `x := y` is an assignment and `b : Boolean := ...` a local variable declaration. Static analysis will ensure that all identifiers exist in the current scope of the class of the swim lane and that everything is type correct w.r.t. the class diagram.

The protocol uses no cryptography because all communications are assumed to be secure against an attacker as specified in the deployment diagram (Fig. 2). The ATM counts how much money it issues with `moneyPaidOut := moneyPaidOut + money`. This is not necessary for the protocol, but needed in order to express the security property.

Smart cards, services, and terminals are very different in reality, but there is almost no difference in their treatment in the activity diagram. This is the idea of model-driven development—the model abstracts from technical details. Only the class and deployment diagrams distinguish the components by applying different stereotypes. However, this is not the complete truth. The modeler must be aware that a smart card has very limited resources and cannot be used to store data in the same manner as a service or terminal.

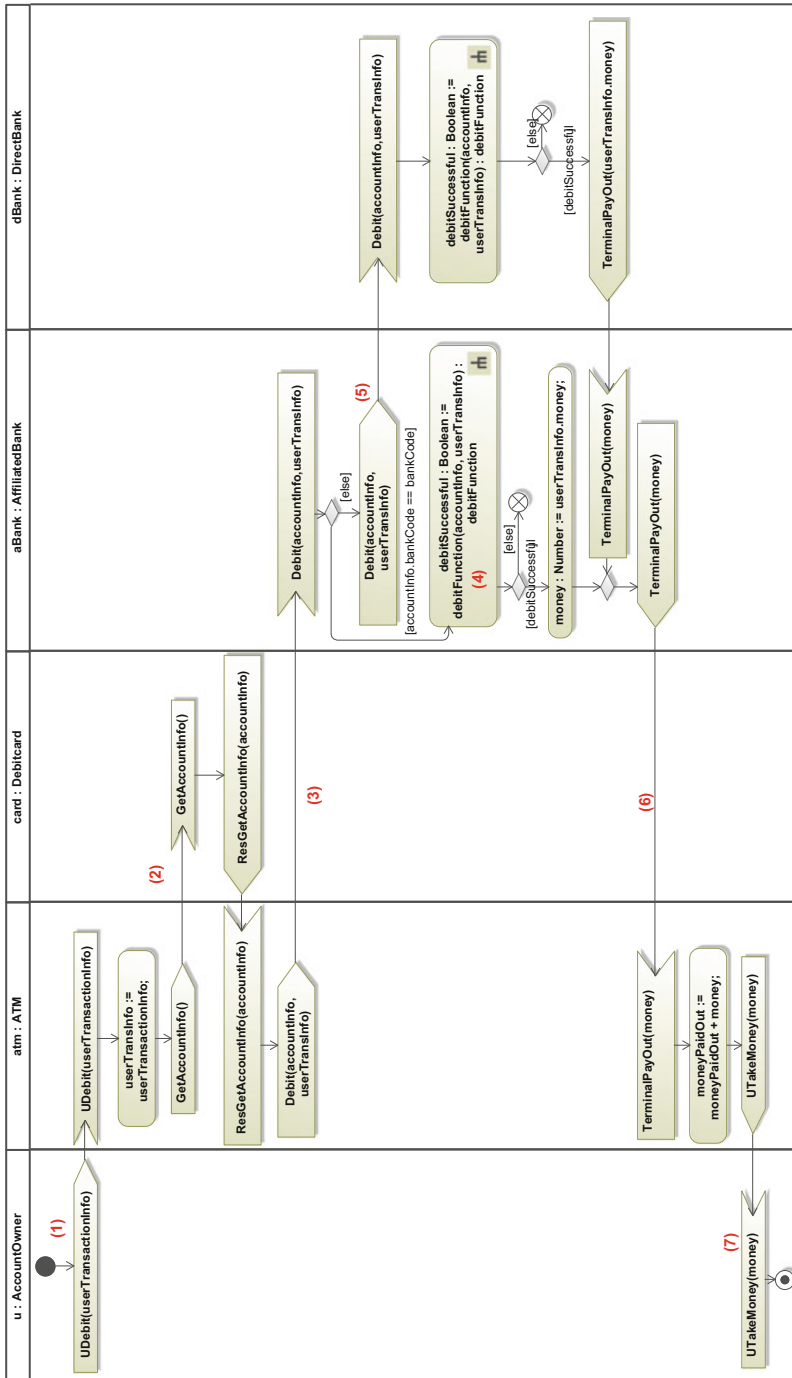


Fig. 4 Protocol to withdraw money from an ATM

3.3.3 Overview over the Handshake Protocol

The other use case in the example is the online transaction. It is much more complicated than using an ATM because first a secure channel must be set up between the smart card and the online banking service by cryptographic means and second because the money transfer to another bank may fail in which case the transaction must be reverted.

In Fig. 5, the handshake protocol is shown as a sequence diagram. A sequence diagram in SecureMDD serves only documentation purposes and shows only the involved components and message types that are exchanged. This makes it impossible to generate code, whereas an activity diagram contains everything needed.

The handshake protocol establishes a secure session between Debitcard and OnlinebankingService using security data types and cryptographic operations defined in SecureMDD. There are four participants involved in this scenario: the AccountOwner, a PC, the Debitcard, and the OnlinebankingService. The user starts the protocol by sending the message UHandshake to the PC (i.e., by interacting with GUI of a program on the PC). The PC begins by sending Handshake to the Debitcard. Afterward, the PC only forwards messages between the card and the online banking service. The card sends a ClientHello containing a nonce encrypted with the public key of the online banking service and the card's certificate. The server answers with ServerHello containing the card's nonce and a server nonce encrypted with the card's public key. As the last major step, the card generates a session key and sends it together with the server nonce encrypted with the server's public key to the online banking

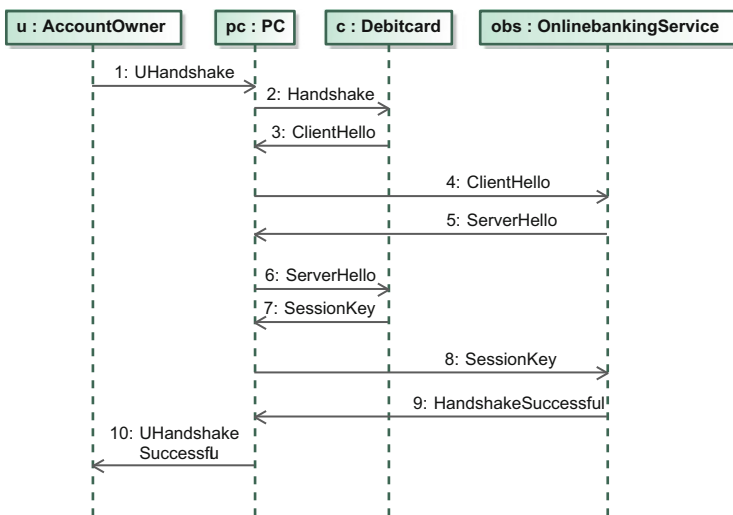


Fig. 5 Messages of the handshake protocol

service. This is a standard challenge-response protocol that is used in a similar form in TLS and many other protocols. Since the protocol has nothing that is specific to services and since cryptographic protocols in SecureMDD have been described in detail elsewhere (e.g., [42, 45, 46]), the activity diagram is omitted here.

3.3.4 Protocol for Money Transfer

Figure 6 shows the protocol to transfer money from one account to another one. It will only run after a successful handshake. This is indicated by a state `AUTHENTICATED` that is checked by the card (2). If the check succeeds, the online transaction can be processed. The account owner has to type in the transaction data, namely, the PIN, the amount of money that should be transferred, and the receiver's account information. This data (`uoti`) is sent from the PC to the card (1). The card checks the state, wraps `uoti` and the account information (`accountInfo`) that is initially stored on the card in the message `TransactionData`, and encrypts it with the previously exchanged session key (3). `encrypt` is a predefined operation in MEL. Then the state is set to `IDLE`, and the encrypted data is sent to the online banking service via the PC. After receiving the message, the service checks that its state is also `AUTHENTICATED` and decrypts the received data with the previously exchanged session key. The state is set back to `IDLE` to avoid replay attacks, and on the basis of the card holder's bank code, it is checked which bank the card holder account belongs to. Depending on this, the message is either sent over the port `affiliated` to the affiliated bank (4) or over the port `direct` to the direct bank (5). The ports are modeled in the deployment diagram (see Fig. 2). If bank `b1` receives a transaction message with PIN, the amount of money that should be transferred, the sender's account information, and the receiver's account information, it checks if the card holder account belongs to it. If so, it checks if the receiver's account also belongs to it and processes the transaction internally (6). Otherwise, the bank deducts the sum from the card holder's account and sends a request to the bank of the receiver's account to increase the sum in its account (7). If this fails because of a nonexistent account number, `b1` is notified, and the deduction from the card holder account is revoked (8). The failure (denoted by the flow `final`) will be propagated back to the user. Otherwise, the transaction was successful, and a message is sent back over the online banking service and the PC to the user. But before the notification is sent to the user, the PC closes the session with the online banking service using the stereotype `«closeSession»` (9). The online banking service must store the state and the session key across several protocol steps. Therefore, the service is stateful and provides a new instance for every invoker. Because the same instance could also be used in another protocol, it is necessary to model the first call of a stateful service. Therefore, the stereotypes `«openSession»` and `«closeSession»` are supported. `«openSession»` was used in the handshake protocol which was not shown.

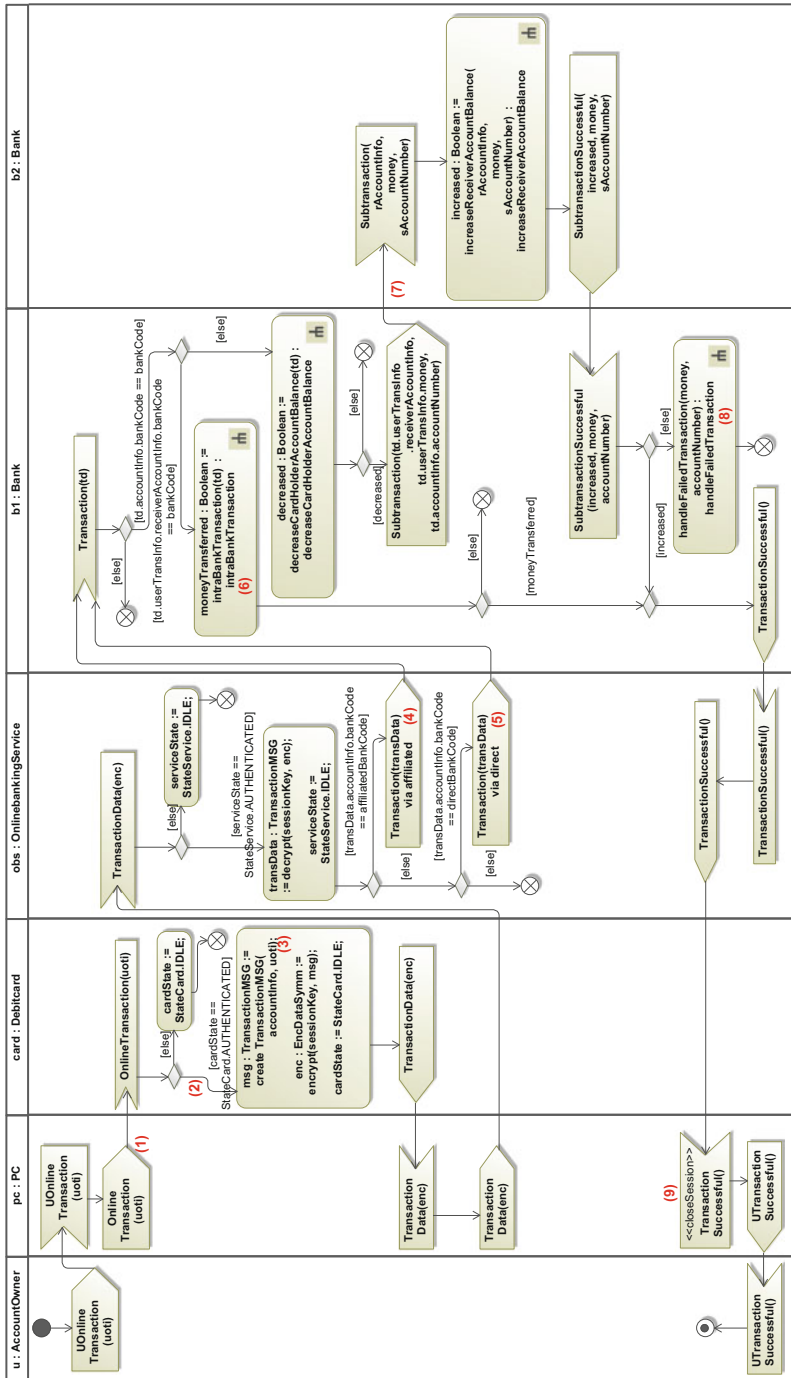


Fig. 6 Protocol to transfer money from one account to another

3.3.5 Structuring Protocols

In Fig. 6, we have seen the use of the superclass Bank in an activity diagram. This is very useful to avoid the modeling of redundant behavior and makes the diagrams clearer. The deployment diagram (see Fig. 2) ensures that if b1 is a DirectBank, then b2 is an AffiliatedBank and vice versa.

Some functionality is encapsulated in methods that are predefined or defined in the model. Methods designed in the model like `intraBankTransaction`, `decreaseCardHolderAccountBalance`, `increaseReceiverAccountBalance`, and `handleFailedTransaction` allow big and complex protocols to be divided into smaller diagrams, so that all of them remain clear.

The activity diagram `increaseReceiverAccountBalance` (Fig. 7) models an internal behavior. Therefore, it has only one swim lane for a Bank. Activity

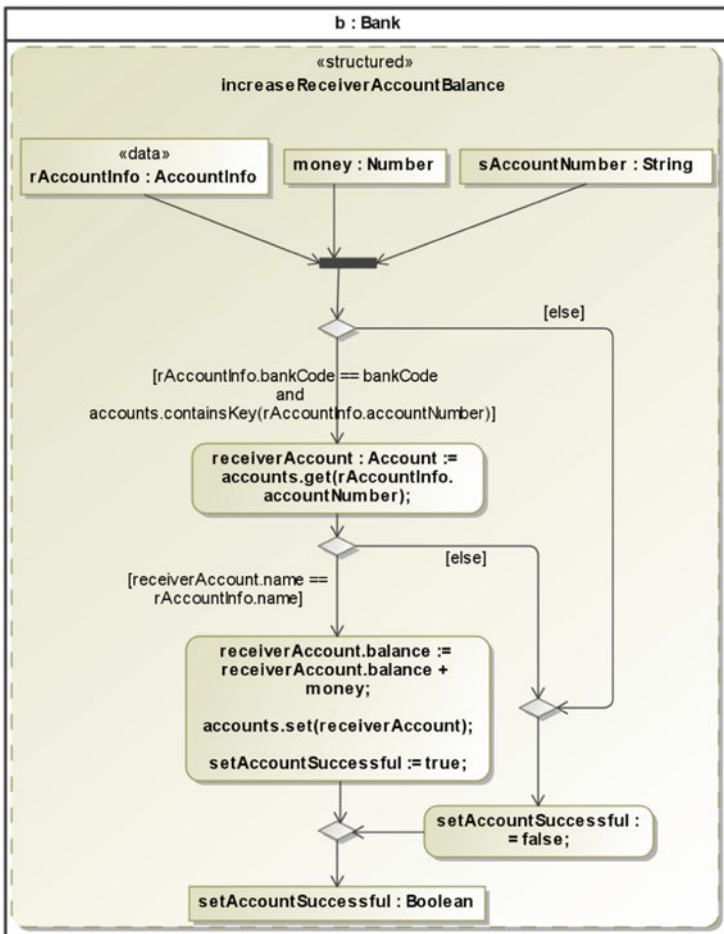


Fig. 7 Sub-activity increaseReceiverAccountBalance

parameters are used to pass arguments. The operation checks if the bank code is correct and if the account number exists. If this is the case, the account is credited and the operation returns true. Otherwise, nothing happens and false is returned. The graphical visualization of the operation is the choice of the modeler. It is also possible to write the whole code as one big blob.

This concludes the description of the example. All diagrams can be found on our Web page.³ It shows how complex applications involving different types of components (services, terminals, PCs, smart cards, and users) can be modeled in SecureMDD. An important aspect is that the full behavior of the application can be modeled, the communication protocol and the internal behavior of the components. SecureMDD provides a UML profile, predefined cryptographic operations and data types, and modeling guidelines to easily model distributed security-critical applications.

The next section describes the formal specification and verification of such applications.

4 Formal Specification and Verification

In this section, the formal model which is automatically generated from the platform-independent UML model is introduced. In Sect. 4.1, an overview of the transformation process is given. Section 4.2 introduces the static part of the formal model, i.e., the data types, components of the application, and message types. In Sect. 4.3, the specification of the dynamic aspects of the system are described with ASMs. Sections 4.4 and 4.5 explain specific details for services, and Sect. 4.6 reports on the formal verification of the example.

4.1 Overview of the Transformation Process

The formal model that is generated from the UML diagrams is an abstract view of the whole UML model (which is a representation of the complete system under development). It contains an arbitrary but finite number of components (smart cards, terminals, and services) and has an arbitrary number of interleaved protocol runs.

The static aspects of an application, i.e., the components, data types, communication infrastructure, and the attacker, are defined using algebraic specifications. The dynamic part of an application, i.e., the cryptographic protocols, is given as an ASM. The ASM consists of two sets of rules: rules defining the behavior of the attacker and rules defining the dynamic behavior of the components (*agents* in the formal model). Executing rules induces a trace of states. Since applicable rules are

³<http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD/>.

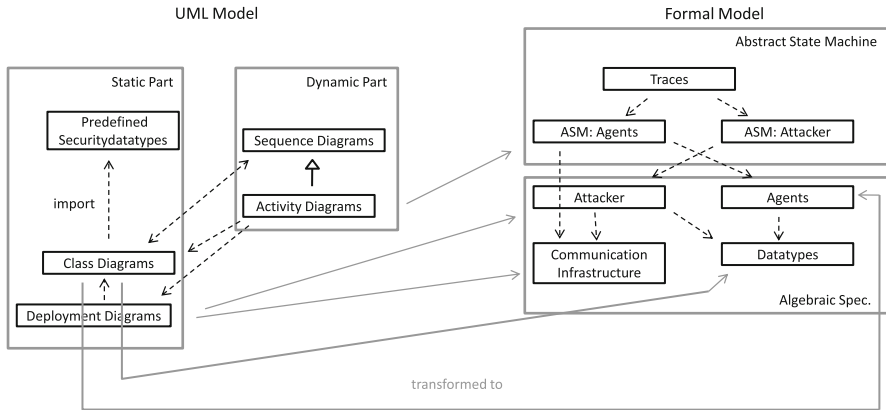


Fig. 8 Dependencies between the UML model and the formal model

chosen nondeterministically, a set of traces is obtained, “everything that can happen in this world”. Figure 8 shows the components of the formal model and from which part of the UML model they are generated. Technically, a UML model is loaded into Eclipse with the Eclipse modeling framework, and specification text suitable for our interactive theorem prover KIV [5] is generated with XPand [61], also part of the Eclipse modeling framework. Other provers could be supported by writing new XPand transformations.

The generated formal model is then used to prove the security of the modeled application with KIV. One relevant security property for the banking example is that the sum of (electronic) money in the system is constant, i.e., no money is lost, and it is not possible to “generate” money. Money dispensed at an ATM is also counted. This can be formulated as an OCL constraint for the class diagram and is translated into a proof obligation in the formal model.

4.2 The Static Part: Data Types and Algebraic Specifications

4.2.1 Data Types

The data of the application, i.e., the messages, predefined security data types, and data types defined in the class diagrams, are translated into algebraic specifications. To reduce the gap between the UML models and the formal model used for interactive verification, we use the same data types as in the class diagrams and do not add a generic data format like some other approaches, e.g., [26, 54]. This

simplifies the verification considerably. For example, an account is specified as a freely generated data type

```
Account = mkAccount( .accountNumber : string;
                    .name : string;
                    .pin : Secret;
                    .balance : int;
                    .dispo : int);
```

and the data type message consists of all 21 messages:

```
message = mkTransactionData( .msg : EncData) with isTransactionData
      | mkTransactionSuccessful with isTransactionSuccessful
      :
      :
```

The specification is not intended to faithfully represent instances of arbitrary class diagrams. Without a heap or references (or something similar), it is not possible to model arbitrary pointer structures like cycles or shared objects. However, this is not necessary since messages and data used in a cryptographic protocol do not (and should not) use such features.

4.2.2 Dynamic Functions

Every instance of a component class (service, terminal, smart card, or user) becomes an agent in the formal specification because it has a behavior as defined by the protocol steps. The attributes of the component classes are modeled as dynamic functions that are modified (updated) by the ASM rules. For each attribute, one dynamic function that maps an agent to the attribute's value is defined. For example,

$ATM\text{-}moneyPaidOut : agent \rightarrow int;$

is the dynamic function for the `moneyPaidOut` attribute of class `ATM`, and `ATM-moneyPaidOut(ag)` returns the value of the attribute for a given `ATM` agent `ag`. The banking system contains 20 dynamic functions. Experience has shown that slicing a class into its attributes simplifies verification as compared to one dynamic function that returns the full state of an agent. The reason is that an update of one attribute and a lookup of another attribute are syntactically disjoint:

```
update(ATM-userTransInfo(ag), uti) and ATM-moneyPaidOut(ag)
as compared to something like
update(ATM(ag), update(ATM(ag).userTransInfo, uti))(ag).moneyPaidOut
```

The specification guarantees that the number of agents is finite so that it is safe to iterate over all agents and summarize `ATM-moneyPaidOut(ag)`, i.e., the sum of all money dispensed at all ATMs.

4.2.3 Cryptography and the Attacker

The predefined security data types do not depend on the concrete application, and the transformation is generic for all applications. For example, encrypted data is specified as a freely generated data type `EncData`:

```
EncData = mkEncData( . .key : SymmKey; . .plain : PlainData);
```

This means the encrypted data contains the key used for encryption. This approach is similar to [54]. This allows to easily specify whether a decryption will succeed or not: The key must be the same as the one used for encryption (private and public key pair for asymmetric encryption). In the specification of the attacker, he/she cannot access the key directly so that the behavior of the cryptographic data types and operations is the same as in reality.

The attacker is only implicitly present in the UML model by «Threat» stereotypes. In the formal specification, the attacker is modeled explicitly as a separate agent that can interact with other components by sending and receiving messages. He/she is associated with a set of data that represents his/her knowledge. If the attacker eavesdrops on a communication path and data is sent over that path, it is analyzed and added to the attacker's knowledge. For example, if the attacker obtains an encrypted message and knows the key, he/she decrypts the message and analyzes its contents to find, e.g., secret PINs. Or, if he/she obtains a key, he/she can decrypt some previously obtained encrypted messages that may contain other keys. Again, this is similar to [54]. Conversely, the attacker can only send messages he/she can construct from his/her knowledge. He/she can encrypt and send a message with a key he/she knows (and he/she must know the data to encrypt) or he/she can simply send a previously obtained message (a replay attack) even if he/she cannot decrypt it. The attacker is not able to decrypt messages without knowing the key, or to generate arbitrary keys, or to guess arbitrary nonces or secrets. This models the fact that it is virtually impossible to find a key or nonce by chance or brute force in the lifetime of the application. This is sometimes called the “perfect cryptography assumption” or “symbolic cryptography” as compared to “computational cryptography”. Non-cryptographic data such as numbers or strings are never a secret and can always be used by the attacker in messages. Therefore, a PIN must be modeled as a secret (1234 in itself is known as a number to everybody, but as a PIN, it should be kept secret).

Only the specifications for predefined security data types and cryptographic operations that are actually used will be generated. The banking system uses symmetric and asymmetric encryption, digital signatures, nonces, and secrets (PINs).

4.3 The Dynamic Part: Abstract State Machine and Traces

The dynamic part of the application, i.e., the security protocols, is defined with activity diagrams in UML. In the formal model, they are translated into an ASM. The ASM consists of a number of rules, basically the individual protocol steps. An applicable rule is chosen nondeterministically in a given state and evaluated yielding another state (one STEP). Rules are applied arbitrarily often (STEP*). In effect, all possible (finite and infinite) traces of the specified “world” are generated:

ASM = STEP*

STEP nondeterministically chooses an action to perform. Possible steps are an attacker step or a step for a component type. For example, if the chosen step is the OnlinebankingService-agent step, an arbitrary component *ag* of the OnlinebankingService class is chosen and the ASM rule ONLINEBANKINGSERVICESTEP is executed for this component.

```
STEP =
  choose asm-step do
    if asm-step = attacker-step then ATTACKER
    else if asm-step = OnlinebankingService-agent-step then
      choose ag with exOnlinebankingService(ag) do
        ONLINEBANKINGSERVICESTEP
    else ...
```

The ONLINEBANKINGSERVICESTEP contains all protocol steps the online banking service can perform. Message passing is modeled with input queues (inboxes for short). A protocol step can be executed if the inbox of a component contains a message of the appropriate type; otherwise, nothing happens.

```
ONLINEBANKINGSERVICESTEP =
  choose port with is-valid-port(ag, port) and inputs(ag)(port) ≠ [] do
  let inmsg = inputs(ag)(port).first in
    inputs := rem(ag, port, inputs) seq
    if (isSessionKey(inmsg)) then SESSIONKEY else
    if (isClientHello(inmsg)) then CLIENTHELLO else
    if (isTransactionData(inmsg)) then TRANSACTIONDATA else
    if (isTransactionSuccessful(inmsg)) then TRANSACTIONSUCCESSFUL else
  ...
```

The message is removed from the inbox, and the correct ASM rule runs. The rule processes the message, performs checks, and updates the state of the component as defined in the corresponding protocol step in the activity diagram. The sending of a new message is modeled by placing it in the inbox of the receiving component. If the attacker can eavesdrop on the communication path (*read* capability in the deployment diagram), the message is also added to the attackers knowledge.

```
TRANSACTIONDATA =
  let enc = inmsg .msg in
    if (serviceState(ag) = AUTHENTICATED) then
```

```

if (not (can_decrypt(sessionKey(ag), enc) and
             isTransactionMSG(decrypt(sessionKey(ag), enc)))) then
    STOPSTEP
else
    let transData = decrypt(sessionKey(ag), enc).transactionMSG in ...

```

TRANSACTIONDATA encapsulates the actual ASM rule. All the intermediate steps are just a convenient grouping of related rules. Aside from some syntactical differences, the abstract program is very similar to the activity diagram. It has the same control structure, checks, and assignments. But there are differences. For example, in MEL, the if statement with the test `if (not (can_decrypt . . .` does not occur. It is added during the transformation. MEL only contains the statement

```
transData : TransactionMSG := decrypt(sessionKey, enc);
```

`decrypt` is a predefined MEL operation that behaves in the generated code (and hence in the real world) as follows: It performs the actual decryption (by applying an algorithm like DES or AES) and then checks that the result is actually an object of the expected type (`TransactionMSG` above). If this is not the case, an exception is thrown. The abstract code has the same behavior. If the key is not correct (in this case `can_decrypt` is false), the result of decryption is a meaningless sequence of bytes (for good algorithms like AES). If the key is correct, the result is something meaningful, but could be of a different type. This is tested in the second part (`isTransactionMSG(decrypt(. . .))`). Since the abstract programming language has no exceptions, the same behavior is obtained with the if-then-else.

One ASM rule is evaluated atomically. We assume that an attacker cannot influence or modify what happens inside a component and cannot read a component's local state. Components are considered as secure in the model. An attacker can only interact with messages and inboxes according to the deployment diagram (Fig. 2, described in Sect. 3.1). Another consequence of this atomicity is that the access to a Web service must be serialized as explained in Sect. 5.2. This can be an efficiency problem but is currently necessary to achieve the same behavior for the model and the generated code.

In case the attacker was chosen instead of a component, the ASM rule for the attacker is called. Then, it is nondeterministically chosen if the attacker suppresses or sends a message. If the attacker suppresses a message, a nonempty inbox is chosen that belongs to a channel where messages can be suppressed by the attacker. Then, a message is deleted from that inbox. In the send case, the attackers generate an arbitrary message from his/her current knowledge. The message is then sent to a randomly chosen inbox accessible by the attacker (i.e., the channel has the attacker-send property).

4.4 *Transport Layer Security and the Attacker*

In Sect. 3, it was mentioned that security stereotypes for connections like «TLS» influence the stereotype «Threat». If an attacker has the abilities to *read*, *send*, and *suppress* messages (i.e., a Dolev–Yao attacker [21] for this connection) and the connection is secured with TLS, the attacker loses some of those abilities. Because TLS is a secure protocol, we use some of its security properties [20]. TLS begins with a handshake that authenticates one or both communication partners. Then messages are encrypted with a session key, their integrity is ensured by a message authentication code (MAC), and a sequence number is used to detect missing or replayed messages. If an error is detected, the connection is closed.

We assume that an attacker is not able to obtain a valid TLS certificate that is accepted by other components. This means he/she cannot initiate a TLS-secured communication if mutual authentication is used. His/her abilities regarding a communication between two components are also limited: The attacker can only read encrypted messages. The used key is a session key exchanged during authentication that will never be used in another session. Therefore, reading the messages is useless for the attacker because he/she cannot decrypt them and they cannot be used for replays because of the sequence number. As mentioned previously, we are only concerned with logical security properties, not traffic analysis where the message length or timing may be important.

Furthermore, in the formal model, the attacker loses his/her ability to send messages, because if the message is not encrypted with the correct session key (which the attacker does not possess), the MAC verification will fail and the message will not be accepted. A replayed message is encrypted with the correct session key, but will not be accepted because of the sequence number. The ability to suppress messages is lost as well because the next message will have an incorrect sequence number. However, the attacker has the ability to terminate the session by replaying a message, because any error in a TLS session leads to termination.

To summarize, it is appropriate to formalize a TLS-secured connection as one where an attacker can either do nothing or can only abort the connection (depending on the annotations in the deployment diagram Fig. 2).

4.5 *Stateful and Stateless Services as Agents*

A service component can be stateless or stateful. Both must be treated slightly different in the formal model. A stateless service is similar to other agents like terminals and smart cards. The formal model may have an arbitrary number of services or it may be restricted to exactly one.

A stateful service however creates an instance of itself for each invoker. The actual code of the invoker calls a manager which is also implemented as a stateless service. It creates an instance of the actual service, deploys it, and returns the address

of this service instance. All this is done by the framework used in the generated code as described in Sect. 5.1. This behavior is not modeled in the UML model of the application, and it is not reflected in the formal specification because it is an implementation issue only. We assume that an attacker has the same abilities for the communication between the two services as between the client and the service. Therefore, no additional security weaknesses are created in the code. Only the address of the service is transmitted which is not a security-critical information leak because either the attacker cannot read this information (if no threat is present in the deployment diagram) or he/she can act as man in the middle anyway (if a threat is present and TLS is not used) or a man-in-the-middle attack is not possible because TLS is used with mutual authentication as described in Sect. 4.4.

For one stateful service, the formal model has an arbitrary number of agents that represent the different new instances of the same service. They all have the same initial attributes that are reset with every connection establishment. Thus, a stateful service is modeled as a set of agents that can be handled as the other agent types.

4.6 Verification of Security Properties

Besides the automatic code generation, the verification of security properties for the generated applications is a major benefit for the development of secure systems. With this approach, we do not have to guess whether the application is secure, since we were able to formally verify it.

Usually only generic properties like secrecy or authentication are proven for security protocols (see [36] for an overview). In contrast, the SecureMDD approach focuses on application-specific security properties [45]. They give better confidence in the properties of the application as a whole. In the banking system (Sect. 3), it is interesting to know that PINs and session keys remain secret, but the real properties are about money. The system has the property that the amount of money is constant in the following sense:

The sum of all account balances plus the amount of all the money that has been withdrawn from cash machines is constant.

This property can be formalized as an OCL constraint that is added to a class in the UML model:

```
Bank.allInstances().accounts.balance->sum() +
ATM.allInstances().moneyPaidOut->sum() = C
```

where C is an unspecified constant. This OCL constraint is translated into a property of the abstract state machine, i.e., the sum is constant in all states of all runs of the ASM. We define

```
BanksMoney(accounts) = Bank.allInstances().accounts.balance->sum()
ATMMoney(moneyPaidOut) = ATM.allInstances().moneyPaidOut->sum()
sum = BanksMoney(accounts) + ATMMoney(moneyPaidOut)
```

`BanksMoney(accounts)` and `ATMMoney(moneyPaidOut)` are the algebraic terms that are the result of the translation of the OCL constraints. `BanksMoney` iterates over the accounts of the banks and summarizes their `balance` attributes, and `ATMMoney` iterates over all components of type `ATM` and sums up their `moneyPaidOut` attributes (see the class diagram Fig. 3).

Then the property can be written formally as

$$\text{init}(\dots) \wedge \text{sum} = C \rightarrow [\text{STEP}^*] \text{sum} = C$$

$[\cdot]$ is the box operator of dynamic logic. The meaning of $[\alpha]\varphi$ is that if program α terminates the condition, φ holds afterward. We start with an initial state (i.e., no protocol steps have been executed, all components are in their initial state, the initial knowledge of the attacker is fixed, and so on). Then, the protocol steps are chosen nondeterministically and executed. Thus, we consider all finite sequences of steps. As described in Sect. 4.3, the ASM consists of a while loop that executes `STEP` or stops. Therefore, the proof works by proving an invariant for every step of the ASM. Of course, the invariant must already hold in the initial state:

$$(\text{INV}(\dots) \wedge \text{sum} = C) \rightarrow [\text{STEP}] (\text{INV}(\dots) \wedge \text{sum} = C)$$

It turns out the property stated above is not quite correct because not yet finished protocol runs must be taken into account. For example, during an online transaction (Fig. 6), there is a situation where one account has been debited, but the receiver not yet credited. In a sense, the money is contained in the message between the two banks—it is in transit—and must be included in the money count. So the actual sum must be computed as follows:

$$\text{sum} = \text{BanksMoney}(\text{accounts}) + \text{ATMMoney}(\text{atms}) + \text{MoneyInTransit}(\text{inboxes})$$

The definition of `MoneyInTransit` uses the inboxes of the ASM that are used to model message passing. The correct OCL constraint therefore must include inboxes. For example, if all inboxes are empty, then `MoneyInTransit` is zero, and we can write

$$\begin{aligned} \text{AllInboxes}().\text{isEmpty} \text{ implies}^4 \\ \text{Bank.allInstances}().\text{accounts.balance} \rightarrow \text{sum}() + \\ \text{ATM.allInstances}().\text{moneyPaidOut} \rightarrow \text{sum}() = C \end{aligned}$$

It is possible to define `MoneyInTransit` in OCL or to specify it directly in the KIV system. As it turns out, the correct definition is quite complicated and was found only after several corrections. The reason for the complexity is twofold: the behavior of the protocol itself and the attacker abilities.

The protocol for withdrawing money from an ATM is quite straightforward: The user's account is debited, and the amount to dispense is sent in a `TerminalPayOut` message (see Fig. 4) either directly to the ATM or via the `AffiliatedBank`. Therefore, we have to count the money in the `TerminalPayOut` messages. The online transfer (Fig. 6) debits the sender's account and sends a

⁴`implies` is an OCL keyword since an arrow `->` is used for operations on collections.

Subtransaction message to the other bank whose money must be counted. The receiving bank answers with a SubtransactionSuccessful message. However, crediting the receiver may fail because of an incorrect account number. In this case, the message contains a false flag, true otherwise. This means the money of a SubtransactionSuccessful message must be counted if and only if the flag is false.

However, the attacker must be taken into account. He/she has very limited capabilities in the example as was explained in Sects. 3.1 and 4.4. He/she has full control over only one communication path, the connection between a PC and the OnlinebankingService (see Fig. 2). This means he/she can inject arbitrary messages, for example, TerminalPayOut, Subtransaction, or SubtransactionSuccessful messages even if this is completely useless (both PC and OnlinebankingService ignore them). Therefore, we must count the money contained in these messages only if they occur between the banks and/or the ATM. There the attacker cannot inject messages, so they must be genuine.

To prove the main security property, a more generalized invariant must be established that consists of several subproperties. This is necessary for almost all applications. In the banking system, the following properties must be established:

- Generic properties about connections between components. Only those communication paths as specified in the deployment diagram are possible (Fig. 2).
- Properties that some messages occur only between dedicated components. This is related to the exact specification of MoneyInTransit and the exact behavior of service answers (because a service always answers its caller).
- Properties about the sending account. The banks are modeled as stateless services. This means that all needed information must be contained in the messages. If a transaction fails, the receiving bank must know the account number of the sending account to initiate a refund (with message SubtransactionSuccessful and flag false). This account number is sent in the Subtransaction message.

Therefore, it must be proved (and the property is actually needed for the main proof) that the sending account number in those two messages really exists.

These properties are far from obvious and were found during failed proof attempts. If it turns out that the invariant is not strong enough (or even incorrect, i.e., not really invariant), it must be modified, and the proofs must be done again. This often happens even if the protocol is secure. The effort can be reduced significantly if it is ensured that the invariant proofs are done automatically by the proof tool. In KIV, this can be ensured with suitable rewrite rules.

The final proof uses 244 lemmas that require 953 user interactions and 17,125 proof steps for their proofs. A little less than half of the effort is needed for the invariant, the rest for properties of the various counting functions. A first version of the case study required 484 theorems, 4,229 user interactions, and 27,038 proof steps. This shows how our verification technique for SecureMDD applications and cryptographic protocols has improved over time. Other case studies show similar

improvements. All specifications, lemmas, and proofs can be found on our Web page⁵ together with several other case studies.

5 Automatic Code Generation

SecureMDD not only supports the modeling and formal verification of security-critical applications but also the automatic generation of runnable code for the application from the class and activity diagrams. As described in Sect. 2, three platforms are supported: smart cards, terminals, and services.

- **Smart cards:** The full smart card code of the application is generated as Java Card code. It can be deployed without any modifications or extensions. Java Card [28] is a version of Java tailored to resource-constrained devices. Java Card has the usual Java statements and expressions, but no strings, no integers, no floats, no threads, no reflection, and no garbage collection. Communication with a smart card is done with sequences of bytes, i.e., byte arrays.

These limitations make programming in Java Card very difficult and error prone. The code generation handles serialization of objects for communication, reuse of objects on the card (this is necessary because of the missing garbage collection), details of short arithmetics, and cryptographic operations. More details can be found in [43].

- **Terminals:** For terminals, the full protocol logic is generated in Java, as well as code for the communication with other components, including object serialization. Only GUI code for interaction with a user must be added. This can be done without modifying or interfering with the protocol logic [23].

The code is similar to hand-programmed Java with one subtle exception. The MEL semantics knows no pointers, but only values because object identities play no role in security protocols. Therefore, the generated code uses value comparison (with `equals` methods) instead of pointer comparison and ensures that assignments behave as if the object was deep cloned. Copying is used only if aliasing and field updates cause incorrect side effects. This can be analyzed given the activity diagrams.

- **Services:** The protocol logic and code for communication and serialization are generated. The code can be deployed without additions or modifications in a Java service framework.

The rest of this section describes the service code generation and deployment in more detail.

Technically, the UML model is loaded into Eclipse with the Eclipse modeling framework. Then model-to-model transformations written in QVT [53] create intermediate platform-specific UML models for each platform. From these models,

⁵<http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD/>.

source code is generated by model-to-text transformations with XPand [61]. Both QVT and XPand are part of the Eclipse modeling framework.

As mentioned in Sect. 2, the transformations are designed so that the generated code is correct and secure with respect to the formal specification. A formal proof that the transformations indeed guarantee this property is an ongoing research.

5.1 Services and Service Communication

The service components are implemented as Java Web services that use SOAP [41] as underlying technology. To implement Web services, Metro⁶ that integrates JAX-WS [47] (Java API for XML—Web Services) is used. JAX-WS is a standard and supports server and clients and can be used by annotated plain old Java objects (POJOs) that are generated by the transformations.

A service is implemented as a Java class that is annotated with `@WebService`, and the public interface of a service is defined by service operations that are annotated with `@WebMethod`. The return value of a service operation is sent to the invoker. For asynchronous message passing, the return value is void and the operation is annotated with `@OneWay`.

A SecureMDD service always contains only one service operation `process` (see Listing 1.) that handles all incoming messages. It invokes the internal method `processMessage` with the message that is wrapped in a `MessageWrapper` object. The wrapping is necessary because JAXB (Java Architecture for XML Binding, part of JAX-WS) needs a container to transmit an object of a class hierarchy with information about the object's run time type. `msg.getMsg()` selects the actual message from the wrapper. This is an object of class `Message`, the common superclass for all messages in the class diagram (Fig. 3). The method `processMethod` makes a case distinction over the actual type of the message and dispatches to one method for each message class. This approach (only one public method to handle all incoming messages) is also used in the terminal and smart card code and behaves exactly as the formal ASM (Sect. 4.3).

```
@WebMethod
public synchronized MessageWrapper process(MessageWrapper msg)
    throws ServiceException {
    MessageWrapper m = null;
    try {
        m = new MessageWrapper(processMessage(msg.getMsg()));
    } catch (java.lang.Exception e) { stop(); }
    return m;
}

private Message processMessage(Message inmsg) throws java.lang.Exception {
    switch (inmsg.getCode()) {
        case Code.DEBIT :
            return processDebit((Debit) inmsg);
    }
}
```

⁶<http://metro.java.net/>.

```

        case Code.TERMINALPAYOUT :
            return processTerminalPayOut((TerminalPayOut) inmsg);
        ...
        default :
            stop();
            return null;
    }
}

```

Listing 1. The single service operation of a SecureMDD service and the dispatcher

The `OnlineBankingService` handles an online transfer by delegating it to the customer's bank. This happens when the service receives a `TransactionData` message (see Fig. 6). For each message, one Java method is generated. The method `processTransactionData` is shown in Listing 2.

```

private Message processTransactionData(TransactionData inmsg)
    throws java.lang.Exception {
synchronized (manager) {
    EncDataSymm enc = inmsg.getMsg();
    if (serviceState == StateService.getAUTHENTICATED()) {
        TransactionMSG transData = (TransactionMSG)
            (EncDataSymm.decrypt(sessionKey, enc));
        setServiceState(StateService.getIDLE());
        if (transData.getAccountInfo().getBankCode()
            .equals(affiliatedBankCode)) {
            return sendMsg(new Transaction(transData), Ports.affiliated);
        } else {
            if (transData.getAccountInfo().getBankCode()
                .equals(directBankCode)) {
                return sendMsg(new Transaction(transData), Ports.direct);
            } else {
                stop();
                return null;
            }
        }
    } else {
        setServiceState(StateService.getIDLE());
        stop();
        return null;
    }
}
}
}

```

Listing 2. Main method for an online transfer

The body of the method is generated from the activity diagram. The method is called with a `TransactionData` object, checks the current state of the (stateful) service, and decrypts the message with the session key exchanged before. The `decrypt` method is part of the SecureMDD implementation of cryptographic operations and raises an exception if the internal decryption does not yield a valid serialized object or if the object is not of the expected type. Both can happen because of an attack. An attacker may send garbage, or a message encrypted with another key, or replay a different message encrypted with the correct key. The ASM rule (in Sect. 4.3) shows the same behavior by corresponding checks in an `if` statement (`if (not (can_decrypt ...))`). An exception always aborts a protocol step, and the ASM rule simply finishes in this case. A generic method `sendMsg` finishes the protocol step by sending the next message (a `Transaction`) to the next service.

A stateful service is implemented by two service classes. One class is annotated with `@Stateless` and the other with `@Stateful`. The first is the *service manager* and the second the actual *stateful service*. An invoker calls the service manager and obtains an address for a fresh instance of the stateful service that was created by the service manager. After that, the invoker communicates with a service instance created exclusively for it (see Sect. 4.5 for a discussion why this does not create a security hole).

Services are invoked by stubs that are automatically generated by the library *wsimport* that is a part of JAX-WS. Stubs manage the communication between client and service by mapping Java objects to XML documents and vice versa as well as the transport of the XML documents. Which stubs have to be generated for a service invoker component is defined by the deployment diagram. The generated stubs also contain the classes that are transferred, but without method implementations. Hence, the classes generated by *wsimport* are replaced by the classes that are generated by the model transformation.

The deployment diagram specifies how many service instances of the same component can be invoked by one component instance. If a component instance can invoke only one instance of a service component (denoted by multiplicity 1), then the stubs are generated at deployment time and can be used without changes. Otherwise (with multiplicity *), for each service to invoke, its address must be known. The stubs that are generated for one instance can be used with an address for any instance of the same type.

5.2 *Parallel Service Invocation*

Because services can access shared memory at the same time, it is important to consider parallelism. Especially, protocol steps with read and write access have to be executed atomically. This is the intended behavior, and the ASM behaves like this. A possible solution is that the developer of an application has to manage the synchronization explicitly in the UML model. A more comfortable way is that the synchronization is managed automatically by the code generation. Our current solution is to execute only one service operation at any given time, i.e., to force a completely sequential behavior. For a stateless service, the bodies of the operations are synchronized on the service instance, and for stateful services, the bodies have to be synchronized on the manager instance. Possible future work is to look for a strategy to synchronize only the critical parts of the code.

5.3 *Transport Layer Security*

TLS with server side authentication is provided by a Java library. If another TLS implementation should be used, the generated TLS code can be disabled. TLS

uses keys and certificates; thus, we need a key and trust store to provide them. The key store contains asymmetric key pairs, while the trust store provides signed certificates; if a certificate represents a certificate authority, all certificates issued by this authority will also be accepted. To use TLS, keys and certificates have to be transferred inside a secure environment before the system can be deployed.

5.4 Deployment

The deployment diagram contains all important information to deploy an application (some additional information is defined in the class diagram). To deploy the banking system, the following information is important. The application model contains a user that represents a real human, a smart card component, and components that can be deployed on PCs or servers. Except for the `OnlinebankingService`, each of these components can be instantiated multiple times, i.e., an arbitrary number of PC applications and Bank services can be deployed. The generated Java Card code can be loaded onto an arbitrary number of smart cards, and the ATM code can be loaded onto an arbitrary number of real ATMs.

The online banking service can be accessed by many account owner PCs at the same time and can invoke operations of the bank services. The communication between a user and a PC has to be secured against an attacker (i.e., against shoulder surfing and malware). A PC can communicate with a service over any kind of network, and services can also communicate over any kind of network.

For the PC, a Java package containing all necessary code is generated. The class PC can be instantiated on any device that supports Java and is secured against an attacker (i.e., free of malware). For the bank service, a Java package is generated as well. This package can be deployed inside a container on any Java Web server.

The class diagram defines attributes that have to be initialized at the start of a service (they are annotated with `«Initialize»` in the class diagram). For the banks, these are the initial accounts and bank codes; for the `OnlinebankingService`, public and private keys and bank codes; etc. These attributes must be passed as parameters to the constructor before the service is started.

SecureMDD provides predefined key-value lists that are used for the accounts. The generated code provides a prototypical implementation that resides in memory. It can be replaced by any other implementations or databases that implement the same interface.

Because requirements usually change during software development, it is useful that code for the modeled applications can be automatically generated and tested. For testing, a test case that initializes all instances, deploys all services, and calls the user messages to execute the protocols has to be written. If such a test case exists, the whole process from code generation over service deployment up to running the test code can be invoked with one click inside Eclipse. In our test framework, all services are deployed on one lightweight server that is integrated in Java, but of course it is possible to deploy the services on other servers.

The full generated and runnable code can be found on our website.⁷

6 Related Work

Related work relevant for this chapter can be divided into three categories: verification of cryptographic protocols, model-driven development of security-critical systems, and model-driven development of service applications. They are treated in turn.

6.1 Verification of Cryptographic Protocols

A lot of verification techniques and tools to prove the security of cryptographic protocols exist; an overview is given in [36]. These techniques can be divided into three categories: belief logics (e.g., [17]), state exploration (e.g., [6, 37]), and theorem proving (e.g., [10, 39, 54]). Most of them are based on automatic tools and focus on generic security protocols (which are not specific to an application, e.g., authentication protocols) and prove standard security properties. In contrast, the protocols of the banking system as well as the security property highly depend on the considered application. It is not clear if automatic tools can cope with that kind of security property.

Some approaches dealing with application-specific security properties exist. One method that is related to ours is the inductive approach of Paulson [54] that uses the theorem prover Isabelle for verification and was successfully applied to several case studies. Bella extends the inductive approach to deal with smart cards [8] but concentrates on generic security protocols as well. In [9], Bella et al. give an overview of their work on SET (Secure Electronic Transaction), a set of e-commerce protocols devised by Visa and MasterCard. The case study is formally modeled and verified using the inductive approach. Considered and proven (application-specific) security properties are that the payment information of the customer are only known to the bank, not to the merchant, and that the order information is not known to the bank.

Another interesting case study with application-specific security properties was the Mondex electronic purse. Mondex has received a lot of attention because its formal verification has been set up as a challenge for verification tools [60] that several groups worked on. The results of the participating groups are summarized in [30]. Mondex is about transferring money (from one card to another), and the main property is that no money is lost.

⁷<http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD/>.

6.2 *Model-Driven Development of Security-Critical Systems*

Surveys can be found in [33] and [29]. The most closely related approach is UMLSec by Jan Jürjens ([32] and newer work). He developed a method to model systems based on cryptographic protocols with UML. Inputs for model checking and automatic theorem proving are automatically generated from the models and standard security properties are proven. Besides several other case studies, Jürjens worked on the security of the Common Electronic Purse Specification (CEPS) for cash-free point-of-sale transactions. One considered security property was that the sum of balances of all smart cards (which are used for payments) and all cards of the merchants (where the earned money is stored) is the same at any time. The appropriate proof was not done by tools but is paper based [31]. UMLSec does not aim at generating runnable code since the focus is on modeling only the security-related parts.

Smith et al. [58] model security protocols with UML class diagrams and state machines. Partial code can be generated from the state machine and tested against attacks. The attacker model and the number of components is fixed. Formal verification is not supported. Bushager et al. [18] use UML use case and sequence diagrams to model smart card protocols. Partial code can be generated, but since the internal behavior of the components is not modeled, that code must be added by hand. Verification is also not supported. There is quite a lot of work on modeling access control. We mention only SecureUML by Basin et al. [7]. UML class diagrams are used to model security-critical applications with role-based access control (RBAC). Specific authorization constraints can be defined with OCL. Code generation is not supported.

6.3 *Model-Driven Development of Service Applications*

The approach developed by Deubler et al. [19] considers the development of security-critical service-oriented systems. For modeling and verification, it uses the tool AUTOFOCUS [27] that provides its own modeling language similar to UML. The considered security mechanisms are authentication and authorization that are proved with a model checker. Application-specific properties as well as code generation are not considered. AUTOFOCUS was used by Grünbauer et al. [25] to model a banking application where a customer can submit a transfer order online. Essentially this is the handshake protocol and the first part of the transfer protocol of our banking system (the actual transfer is not considered). The confidentiality and authenticity of the order is proved with a model checker. However, the attacker capabilities had to be simplified because the original model was too complex for automatic verification.

SECTISSIMO by Memon et al. [40] is a framework to model security-critical services with a business process language, enrich the model with security policies,

and generate code to enforce the policy. The security policy can be composed of a set of given cryptographic primitives and protocols. Formal verification of security properties is not supported.

MDD4SOA developed by Mayer et al. [38] is a model-driven approach for service orchestration that transforms a platform-independent model into several PSMs and those to partial code for the languages BPEL, WSDL, and Java and the formal language Jolie. It uses its own UML profile [22] to allow the modeling of SOA and verify properties with the formal language Jolie. Compliance of service orchestration with their interaction protocols can be checked automatically [56]. Security aspects are not considered, and the full behavior of the components is not modeled.

There is some work on model-driven development of Web services in general. Baina et al. [4] use UML state machines to describe service communication and generate BPEL-based service skeletons that implement conversation management logic. Gronmo et al. [24] import Web service descriptions into UML, composite them, and generate a new Web service description. Both approaches focus on service composition and neither model the complete service behavior nor consider security. Sheng et al. [57] focus on context-aware Web services. UWE4JSF by Kroiss et al. [35] defines a UML profile and can generate executable code.

The following papers consider security in a model-driven approach for Web service architectures. Nakamura et al. [51] describe security with UML by primitives that will be transformed into security configurations such as WS-SecurityPolicy. They do not verify the security of an application and only parts of an application are modeled. Alam et al. [1] use OCL and role-based access control (RBAC) and generate eXtensible Access Control Markup Language (XACML) policy files to define a security infrastructure. This approach focuses on access control only.

Formal approaches to services can also be used to formally verify security properties. SecureMDD uses ASMs to define the behavior of components like services, and if services communicate with other services, this is a simple form of service orchestration. More elaborate approaches to specify services with ASMs by Börger and Thalheim [16] and Börger and Sörensen [14] contain interesting ideas for future extensions of SecureMDD.

We are not aware of an approach like ours that allows model-driven development of security-critical Web service applications, generates executable code, and guarantees application-specific security properties for the modeled system by using interactive verification.

7 Conclusion

We presented in detail how services are supported in our SecureMDD approach. Security in service applications is an important issue. Since it is difficult to express business security requirements with standard security properties, our approach supports the consideration of application-specific security properties in all stages

of the development process. Secure applications using smart cards, terminals, and services are modeled with extended UML and the domain-specific language MEL. The banking system used as a case study demonstrates the need to verify application-specific security properties even if standard security protocols like TLS are used. From a platform-independent UML model, runnable code as well as a formal specification is generated automatically. The code describes an application in full detail and can be deployed and used without any changes. The formal specification is used to verify the application-specific security properties. The code is correct and secure with respect to the formal specification.

Future work for services includes handling parallelism in a more efficient way, integrating a real database and WS-Security standards, and extending the communication structure to allow more complex service orchestrations. SecureMDD will also support Android and Apps as an additional platform in the future.

References

1. Alam, M.M., Breu, R., Breu, M.: Model driven security for web services (MDS4WS). In: 8th International Multitopic Conference, 2004. Proceedings of INMIC 2004, pp. 498–505. IEEE, Piscataway (2004)
2. Anderson, R.J., Needham, R.M.: Programming satan’s computer. In: *Computer Science Today*, vol. 1000, pp. 426–440. Springer, Heidelberg (1995)
3. Armando, A., Arzac, W., Avanesov, T., Barletta, M., Calvi, A., Cappai, A., Carbone, R., Chevalier, Y., Compagna, L., Cúellar, J., et al.: The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In: Proceedings of TACAS 2012 – Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 7214. Springer, Heidelberg (2012)
4. Baina, K., Benatallah, B., Casati, F., Toumani, F.: Model-driven web service development. In: *Advanced Information Systems Engineering*, pp. 527–543. Springer, Heidelberg (2004)
5. Balsler, M., Reif, W., Schellhorn, G., Stenzel, K., Thums, A.: Formal system development with KIV. In: *Fundamental Approaches to Software Engineering*. Lecture Notes in Computer Science, vol. 1783. Springer, Heidelberg (2000)
6. Basin, D.A., Mödersheim, S., Viganò, L.: OFMC: a symbolic model checker for security protocols. *Int. J. Inf. Secur.* **4**(3), 181–208 (2005)
7. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.* **15**, 39–91 (2006)
8. Bella, G.: Mechanising a protocol for smart cards. In: Proceedings of e-Smart 2001, International Conference on Research in Smart Cards. Lecture Notes in Computer Science, vol. 2140. Springer, Heidelberg (2001)
9. Bella, G., Massacci, F., Paulson, L.C.: Verifying the SET purchase protocols. *J. Automat. Reas.* **36**(1–2), 5–37 (2006)
10. Blanchet, B.: Automatic verification of correspondences for security protocols. *J. Comput. Secur.* **17**(4), 363–434 (2009)
11. Borek, M., Moebius, N., Stenzel, K., Reif, W.: Model-driven development of secure service applications. In: 2012 35th Annual IEEE Software Engineering Workshop (SEW), pp. 62–71. IEEE, Piscataway (2012)
12. Borek, M., Moebius, N., Stenzel, K., Reif, W.: Model checking of security-critical applications in a model driven approach. In: *Software Engineering and Formal Methods*. Springer, Heidelberg (2013)

13. Borek, M., Moebius, N., Stenzel, K., Reif, W.: Security requirements formalized with ocl in a model-driven approach. In: 2013 IEEE Model-Driven Requirements Engineering Workshop (MoDRE). IEEE, Piscataway (2013)
14. Börger, E., Sörensen, O.: BPMN core modeling concepts: inheritance-based execution semantics. In: Handbook of Conceptual Modeling. Theory, Practice, and Research Challenges, pp. 287–332. Springer, Heidelberg (2011)
15. Börger, E., Stärk, R.F.: Abstract State Machines—A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
16. Börger, E., Thalheim, B.: Modeling workflows, interaction patterns, web services and business processes: the ASM-based approach. In: Proceedings of ABZ 2008. Lecture Notes in Computer Science, vol. 5238. Springer, Heidelberg (2008)
17. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. *ACM Trans. Comput. Syst.* **8**(1), 18–36 (1990)
18. Bushager, A., Zwolinski, M.: Modelling smart card security protocols in systemC TLM. In: IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing, pp. 637–643. IEEE Computer Society, Piscataway (2010)
19. Deubler, M., Grünbauer, J., Jürjens, J., Wimmel, G.: Sound development of secure service-based systems. In: Proceedings of the 2nd International Conference on Service Oriented Computing, pp. 115–124. ACM, New York (2004)
20. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.2. IETF Network Working Group. <http://www.ietf.org/rfc/rfc5246.txt> (2008)
21. Dolev, D., Yao, A.C.: On the security of public key protocols. In: Proceedings of 22th IEEE Symposium on Foundations of Computer Science. IEEE, Piscataway (1981)
22. Foster, H., Gönczy, L., Koch, N., Mayer, P., Montangero, C., Varró, D.: UML extensions for service-oriented systems. In: Rigorous Software Engineering for Service-Oriented Systems, pp. 35–60. Springer, Heidelberg (2011)
23. Grandy, H., Stenzel, K., Reif, W.: Object-oriented verification kernels for secure Java applications. In: Aichering, B., Beckert, B. (eds.) SEFM 2005 – 3rd IEEE International Conference on Software Engineering and Formal Methods. IEEE, Piscataway (2005)
24. Gronmo, R., Skogan, D., Solheim, I., Oldevik, J.: Model-driven web services development. In: 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2004. IEEE'04, pp. 42–45. IEEE, Piscataway (2004)
25. Grünbauer, J., Hollmann, H., Jürjens, J., Wimmel, G.: Modelling and verification of layered security protocols: a bank application. In: Proceedings of SAFECOMP 2003. Lecture Notes in Computer Science, vol. 2788. Springer, Heidelberg (2003)
26. Haneberg, D., Grandy, H., Reif, W., Schellhorn, G.: Verifying smart card applications: an ASM approach. In: International Conference on integrated Formal Methods (iFM) 2007. Lecture Notes in Computer Science, vol. 4591. Springer, Heidelberg (2007)
27. Huber, F., Molterer, S., Rausch, A., Schatz, B., Sihling, M., Slotosch, O.: Tool supported specification and simulation of distributed systems. In: Proceedings, International Symposium on Software Engineering for Parallel and Distributed Systems, 1998, pp. 155–164. IEEE, Piscataway (1998)
28. Java Card 2.2.2 Application Programming Interfaces: <http://www.oracle.com/technetwork/java/javacard/specs-138637.html> (2006)
29. Jensen, J., Jaatun, M.G.: Security in model driven development: a survey. In: Sixth International Conference on Availability, Reliability and Security, ARES 2011. Lecture Notes in Computer Science, pp. 704–709. Springer, Heidelberg (2011)
30. Jones, C., Woodcock, J. (eds.): *Form. Asp. Comput.* **20**(1) (2008)
31. Jürjens, J.: Developing high-assurance secure systems with UML: a smartcard-based purchase protocol. In: IEEE International Symposium on High Assurance Systems Engineering. IEEE, Piscataway (2004)
32. Jürjens, J.: *Secure Systems Development with UML*. Springer, Heidelberg (2005)
33. Kasal, K., Heurix, J., Neubauer, T.: Model-driven development meets security: an evaluation of current approaches. In: 44th Hawaii International Conference on System Sciences (HICSS), pp. 1–9. IEEE Computer Society, Piscataway (2011)

34. Katkalov, K., Moebius, N., Stenzel, K., Borek, M., Reif, W.: Model-driven testing of security protocols with secureMDD. In: Fifth IFIP International Conference on New Technologies, Mobility and Security (NTMS 2012). IEEE, Piscataway (2012)
35. Kroiss, C., Koch, N., Knapp, A.: UWE4JSF: a model-driven generation approach for web applications. In: 3rd Workshop on The Web and Requirements Engineering at ICWE 2012. Lecture Notes in Computer Science, vol. 5648, pp. 493–496. Springer, Heidelberg (2009)
36. Lopez Pimental, J.C., Monroy, R.: Formal support to security protocol development: a survey. *Computacion y Sistemas* **12**(1), 89–108 (2008)
37. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
38. Mayer, P., Schroeder, A., Koch, N.: MDD4SOA: model-driven service orchestration. In: Proceedings of 12th IEEE International EDOC Conference (EDOC 2008). IEEE, Piscataway (2008)
39. Meadows, C.: The NRL protocol analyzer: an overview. *J. Logic Program.* **26**(2), 113–131 (1996)
40. Memon, M., Hafner, M., Breu, R.: SECTISSIMO: a platform-independent framework for security services. In: Proceedings of the First International Modeling Security Workshop. CEUR Workshop Proceedings, vol. 413. <http://ceur-ws.org/Vol-413/> (2008)
41. Mitra, N., Lafon, Y.: SOAP Version 1.2. W3C (2007)
42. Moebius, N., Stenzel, K., Reif, W.: Modeling security-critical applications with UML in the SecureMDD approach. *Int. J. Adv. Softw.* **1**(1), 59–79 (2008)
43. Moebius, N., Stenzel, K., Grandy, H., Reif, W.: Model-driven code generation for secure smart card applications. In: 20th Australian Software Engineering Conference. IEEE, Piscataway (2009)
44. Moebius, N., Stenzel, K., Grandy, H., Reif, W.: SecureMDD: a model-driven development method for secure smart card applications. In: Workshop on Secure Software Engineering, SecSE, at ARES 2009. IEEE, Piscataway (2009)
45. Moebius, N., Stenzel, K., Reif, W.: Formal verification of application-specific security properties in a model-driven approach. In: Proceedings of ESSoS 2010 - International Symposium on Engineering Secure Software and Systems. Lecture Notes in Computer Science, vol. 5965. Springer, Heidelberg (2010)
46. Moebius, N., Stenzel, K., Borek, M., Reif, W.: Incremental development of large, secure smart card applications. In: Proceedings of the Workshop on Model-Driven Security. ACM, New York (2012)
47. Mordani, R., Chinnici, R., Hadley, M.: The Java API for XML-Based Web Services (JAX-WS) 2.0. JCP (2006)
48. Murdoch, S.J., Drimer, S., Anderson, R., Bond, M.: Chip and PIN is broken. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 433–446. IEEE, Piscataway (2010)
49. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web Services Security: SOAP Message Security 1.0. OASIS (2004)
50. Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., Granqvist, H.: WS-SecurityPolicy 1.2. OASIS (2006)
51. Nakamura, Y., Tsubori, M., Imamura, T., Ono, K.: Model-driven security based on a web services security architecture. In: IEEE International Conference on Services Computing, pp. 7–15. IEEE, Piscataway (2005)
52. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. *Commun. ACM* **21**(12), 993–999 (1978)
53. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1. <http://www.omg.org/spec/QVT/1.1/> (2011)
54. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.* **6**, 85–128 (1998)
55. Ray, M., Dispensa, S.: Renegotiating TLS. Technical Report, PhoneFactor Inc. (2009)

56. Schroeder, A., Mayer, P.: Verifying interaction protocol compliance of service orchestrations. In: Proceedings of the 6th International Conference on Service-Oriented Computing. Lecture Notes in Computer Science, vol. 5364. Springer, Heidelberg (2008)
57. Sheng, Q.Z., Benatallah, B.: Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In: International Conference on Mobile Business, 2005. ICMB 2005, pp. 206–212. IEEE, Piscataway (2005)
58. Smith, S., Beaulieu, A., Greg Phillips, W.: Modeling and verifying security protocols using UML 2. In: International Systems Conference (SysCon), pp. 72–79. IEEE Computer Society, Piscataway (2011)
59. Stenzel, K., Moebius, N., Reif, W.: Formal verification of QVT transformations for code generation. In: 14th International Conference on Model Driven Engineering Languages and Systems, MODELS 2011. Lecture Notes in Computer Science, vol. 6981. Springer, Heidelberg (2011)
60. Woodcock, J.: First steps in the verified software grand challenge. *IEEE Comput.* **39**(10), 57–64 (2006)
61. Xpand: <http://projects.eclipse.org/projects/modeling.m2t.xpand> (2009)

A Formal Model of Client-Cloud Interaction

Károly Bósa, Roxana-Maria Holom, and Mircea Boris Vleju

Abstract In our former work, we have showed that cloud computing still requires lots of fundamental research. Among many other existing problems in cloud computing, we identified the lack of client orientation and lack of formal foundations as serious deficiencies. In this chapter, we give a summary on our research and discuss the architectures as well as the formal models of some software solutions with which we are going to address (a part of) these two problems in cloud computing.

The solution we propose is a novel and uniform client-cloud interaction approach by which cloud service owners, who may be different from the cloud providers, are able to fully control the usage of their services in the case of each user subscription. In this context, any cloud service can be invoked by distinct devices; therefore, the content must be adapted to various channels and end devices, in particular with respect to needs arising from mobile clients. For a quick and seamless integration between the cloud provider's identity management system and the system used by the client, we introduce the concept of a client-centric tool. An extension of the client-cloud interaction model enables client-to-client interaction (CTCI) in an almost direct way, so that the involvement of cloud services is transparent to the users.

In this chapter, we propose a formalization of this solution that incorporates the major advantages of *abstract state machines (ASMs)* and *ambient calculus* by specifying the algorithms of executable components (agents) in terms of ASMs and by describing their communication topology, locality, and mobility in the terms of ambient calculus.

1 Introduction

Nowadays “cloud computing” is one of the most often used buzzword in computing, and many providers (Amazon, Google, Microsoft, IBM, etc.) of cloud services

K. Bósa (✉) • R.-M. Holom • M.B. Vleju
Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University
Linz, Softwarepark 21, 4232 Hagenberg, Austria
e-mail: k.bosa@cdcc.faw.jku.at; r.holom@cdcc.faw.jku.at; b.vleju@cdcc.faw.jku.at

(infrastructure as a Service (IaaS), software as a service (SaaS), platform as a service (PaaS), data as a service (DaaS), etc.) emphasize the many benefits of outsourcing applications into a (private or public) cloud. In other words, it is suggested that cloud computing represents a mature technology that is ready to be massively used. But it is our conviction that cloud computing still requires lots of fundamental research [99].

In particular, most of the offerings in cloud computing are provider centric. For instance, a client (or tenant) may rent a certain piece of infrastructure, load and execute a piece of software on it, pay for the use, and leave the cloud without leaving permanent traces. Certainly, there are many computing-intensive applications, for example, Web crawling, image processing, machine learning, etc., that fit well into such a scenario. However, if we think of a multiuser database application, its usefulness decreases significantly.

Among many other problems in cloud computing, we identified the lack of client orientation as a serious problem that needs to be addressed in research.¹ This subsumes the problems of identity of tenants, access rights, adaptivity to the needs of clients, and more. For instance, in many cases (e.g., multiuser database applications), it would be indispensable to keep knowledge of users and their rights in the authority of the client instead of in the cloud. The immediate consequence of such an approach is that cloud applications should become hybrid and distributed, as parts of data and software will reside on premise, while others reside off-site in the cloud.

There is another serious lack of formal foundations in cloud computing starting from the simple fact that key notions such as service are not defined. Therefore, in our research, we deal with the challenges in cloud computing that require solutions with more stronger client-side orientation, and we also address the fundamental research question on how a uniform formal model for clouds must look like without any bias to particular languages and technology.

In this chapter, we present the high-level formal models of our cloud-related algorithms and software solutions which have strong client-side aspects and which are integrated into a single cloud service architecture. We split the specification into three parts.

The first part lays out a cloud service infrastructure that provides a transparent and uniform way to interact with its clients (service owners and end users). This includes the following:

- The end users are able to access and combine the available functions of cloud services.
- The owners of the cloud services, who may be different from the cloud providers and who may possess exclusive access to certain cloud resources (service functionalities or data) which is shared among their end users, are able to define

¹We define the term *client* as being a small and medium enterprise (SME) that contracts and uses any cloud service. Similarly, we refer to a *user* as an identity within the SME using a cloud-based service.

some special kind of action schemas called *service plots*, which may be specific for end-user subscriptions, respectively. By these action schemas, clients are able to restrict not only which are the permitted actions belonging to certain cloud resources (e.g., services) for certain end users, but they are also able to specify and tune precisely which are the permitted combinations of these actions to perform certain tasks.

- It is also described how we extended the formal model of the proposed cloud system with a *client-to-client interaction (CTCI)* mechanism via a cloud architecture. The discussed solution for transparent use of services is a kind of switching service, where registered cloud users communicate with each other, and the only role the cloud plays is to switch resources from one client to another.

With respect to identity management in a cloud-based approach, a problem arises in maintaining identity data across the providers. The second component of our system provides a client-focused identity meta-system based on the concept of ASMs, which allows a client to maintain a private identity directory while offering individual users automatic authentication to cloud-based services. In this part, we introduce the concept of an *identity management machine (IdMM)*, an ASM-based, client-centric, single sign-on tool, which deals with the client-side aspects of identity and access management in cloud computing.

The last but not least, the third part of our specification uses ASM ground models for presenting in a rigorous way the proposed *Web application (WA)*, which tries to solve the problem of adaptivity by including aspects regarding content adaptation and displaying. We chose to use ASMs since they permit to design and analyze asynchronous multiple-agent distributed systems, as the cloud architecture we deal with; moreover, thanks to refinement mechanism, we show how ground models can be refined to pseudo-code-like descriptions. A future refinement can possibly bring to the implementation. ASM method also provides a high-level notation that permits to concisely describe complex systems and that can be easily understood by all the stakeholders [25].

As it is shown in Sect. 3, these novel interaction solutions are dynamically reconfigurable, and they can either take place on a cloud or can be easily shifted to the client side and wrapped into a middleware software which may be located in the authority of one or more clients. The latter case can also provide enhanced privacy protection, since the cloud provider does not necessarily know the (real) identities of the end users.

In order to achieve this required flexibility of the model, the component formalization was done in terms of ambient abstract state machines [26, 31], which inherently makes possible to create such dynamically variable architectures. This method incorporates the major advantages of the *abstract state machines (ASMs)* [26, 64] and of *ambient calculus* [38, 61]. Namely, one can describe formal models of complex dynamically reconfigurable distributed (or cloud) systems including mobile components in two abstraction layers such that while the algorithms of executable components (agents) are specified in terms of ASMs, their communication topology, locality, and mobility are described with the terms of

ambient calculus in our method. The abstracted formal architecture presented in this chapter is also going to be served as a framework which can be enriched with other novel client-centric mechanisms in the future.

The rest of the chapter is organized as follows: Section 2 gives an overview on the basic notions of cloud computing and on access control techniques for cloud. Section 3 informally summarizes the components of our integrated cloud architecture. Section 4 introduces the applied formal approaches and gives a short overview on ambient calculus and ambient ASM as well as defines some nonbasic ambient capability actions which are applied in the latter sections. Section 5 describes our high-level ambient ASM model of our cloud service architecture which is equipped with service plots and client-to-client interaction via cloud. Section 6 discusses the specification of client-centric identity and access management for cloud services. Section 7 deals with the ASM ground models of the software components for cloud service adaptivity. Section 8 highlights some aspects of verification of the formal model of our integrated cloud system, which we are going to perform in the near future. Finally, Sect. 9 discusses the related work, and Sect. 10 concludes this chapter.

2 Cloud Computing and Access Control Techniques for Cloud

In this chapter, we give a short overview on the basic notions of cloud computing and on various cloud-related access control techniques in order to make this chapter more self-containing as well as to facilitate its understanding for a wider audience.

2.1 Cloud Computing

Cloud computing appeared years ago as a buzzword in computing technology, being related to other existing technologies, like grid computing and seen as a scalable external data center [114]. Even though people are referring to cloud computing since some time and many providers (Amazon, Rackspace, Google, Microsoft, IBM, etc.) are suggesting that it's a mature technology, we recognize cloud computing as still an evolving paradigm [80].

The *National Institute of Standards and Technology (NIST)* is mentioning important aspects of cloud computing in their definition: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models" [80].

Further on, they mention the cloud computing characteristics: *on-demand self-service*, a consumer can automatically provision computing capabilities without the need for human interaction; *broad network access*, services are available over the network and accessed through a variety of mechanisms (e.g., mobile phones, tablets, laptops, and workstations); *resource pooling*, a provider's capability to use a multi-tenant model for pooling computing resources in order to handle multiple clients; *rapid elasticity*, the possibility to rapidly scale (in and out) resources; and *measured service*, providers are using resources monitoring in order to optimize their services and support the pay-per-use business model. Specific measurements are used, fitting the type of service (e.g., storage, processing, bandwidth, and active user accounts).

Cloud providers offer their services (as a utility) conforming to the following models: *software as a service (SaaS)*, *platform as a service (PaaS)*, and *infrastructure as a service (IaaS)*. SaaS is a service model where a client uses applications offered by the provider that run on a cloud infrastructure. Examples of this service comprise Google Apps for Business [57] and Microsoft Office 365 [81]. PaaS offers the client the possibility to deploy on a cloud infrastructure his or her applications created using predefined languages, libraries, services, and tools that are supported by the cloud provider. Microsoft Azure [15] and Google App Engine [60] are two examples of such services. IaaS is a service model, which provides the client fundamental computing resources, like processing, storage, and networks. In this case, the client can also deploy operating systems, which in the case of PaaS is not possible. An example of such a service model is the Amazon Elastic Compute Cloud (Amazon EC2) [10].

There are also different deployment models of a cloud: private cloud (internal data centers designed for a single organization), community cloud (used by several organizations with shared interests), public cloud (made available for the general public), and hybrid cloud (a combination of the all previous models: some parts of the infrastructure are usable for the general public, while others are ready to use only for some organizations).

Some of the disadvantages that come together with the adoption of cloud services are the following [46, 52]: loss of governance, provider lock-in, isolation failure, and security and privacy issues (insecure interfaces and APIs, malicious insiders, shared technology issues, data loss or leakage, account or service hijacking, unknown risk profile).

2.2 Access Management in Cloud Computing

A detailed description of identity and access management in cloud computing can be found in [109]. The author describes four distinct identity and access management scenarios within the domain of cloud computing. The first scenario entails a traditional model where the client maintains a local identity and access management system which is mirrored on the cloud provider. Such a model is useful only in IaaS where the client has the ability to install custom software on his or her

virtual machine. The second scenario entails a trusted model where the identity information is shared to a cloud provider via the use of previously agreed-upon identity federation protocols. In an identity provider scenario, the client either acts or uses an external identity provider which makes use of common identity federation protocols (such as OpenID, OAuth, SAML) to offer the identity-related information to cloud services. Finally, the author of [109] describes a fourth model where the client makes use of a cloud provider's identity and access management system. This system of *all in the cloud* means that the client has no control over the data and must subscribe to a provider's identity and access management system. Using multiple services across multiple providers further increases the task of maintaining the correct identity information data.

The authors of [63] provide a review of identity and access management in cloud computing. They provide a description of identity and access management as well as a description of existing tools for identity and access management. They then provide a review of the existing identity and access management tools in cloud computing. This review is extended by [83] where the authors better described the technologies and protocols used in identity and access management emphasizing the protocols for identity federation. For small and medium enterprises, [79] describes the challenges when adopting cloud computing services. The authors provide a detailed description of identity and access management in general, also emphasizing the existing identity federation and open standards.

While the authors of [63, 79, 83] emphasize the use of identity federation and existing open standards, it must be noted that most cloud providers tend to use the "all in the cloud" approach when it comes to their identity and access management systems. As such, most providers will offer their own identity system, often not providing the option of using identity federation protocols such as OpenID or OAuth. When such protocols are offered, the providers often act just as identity providers and not relaying parties [50].

3 Overview of the Client-Cloud Interaction Software System

Roughly the formal model of cloud systems employed by us can be regarded as a pool of resources equipped with some infrastructure services. Depending whether these abstract resources represent only physical hardware and virtual resources or the entire computing platforms, the model can be an abstraction of IaaS or PaaS, respectively. The basic hardware (and software) infrastructure is owned by the cloud provider, whereas the software running on the resources may be either rented or owned by some tenants of the cloud (service owners). We assume that these software products may in turn be offered as *services* (denoted by $S_1 \dots S_i$ in Fig. 1) and thus used by some groups of cloud users.

Accordingly, we apply a loose definition of the term service cloud here, where an entity who is different from the cloud provider and who has disposal of the access rights to some hardware or software resources running on the cloud may become

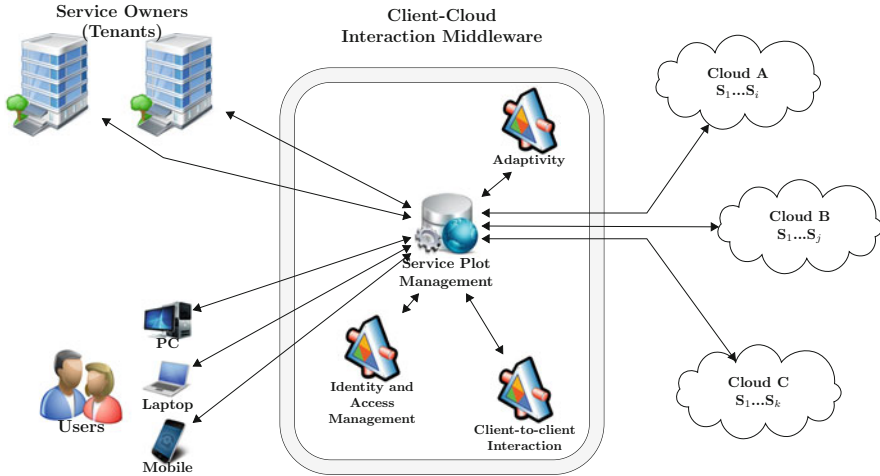


Fig. 1 Architecture

a cloud service provider. Thus, from this aspect, the model can be regarded as an abstraction of a mixture of SaaS and of IaaS (or a mixture of SaaS and PaaS).

We make a distinction between cloud *service owners* (or tenants) and cloud *users* (or end users); see Fig. 1. Users are registered in the cloud, and they subscribe to and use some (software) services available in the cloud. Service owners are usually *small and medium enterprises (SMEs)* that contract with cloud providers, rent some cloud resources, and/or bring and deploy their own resources (e.g., software service instances, data, etc.) on the cloud which they share among their end users. Of course, service owners can also act as normal users, which means they can use services provided by other tenants.

A developed prototype of this cloud service system has been deployed for testing purposes on Windows virtual machines under OpenStack cloud infrastructure software on an IBM CloudBurst server machine.

Due to the applied ambient concept, the relocation of the system component is trivial, and we can apply our model according to different scenarios (see Fig. 2a, b). In our developed prototype and in our case studies (see Sect. 3.4), all our novel software solutions discussed in this chapter are integrated into a compound software component on the client side called the *client-cloud interaction middleware*, which takes place among the end users, the service owners, and the cloud(s) in order to manage the interactions between them; see Fig. 1. In this middleware-based architecture, the various components (e.g., service plot management, client-to-client interaction feature, identity and access management, and content adaptivity) are only loosely coupled with each other; any of them can be eliminated and the others still remain functional.

One of the advantages of the employed middleware-based configuration is that the same instance of the infrastructure services is able to extend the functionalities

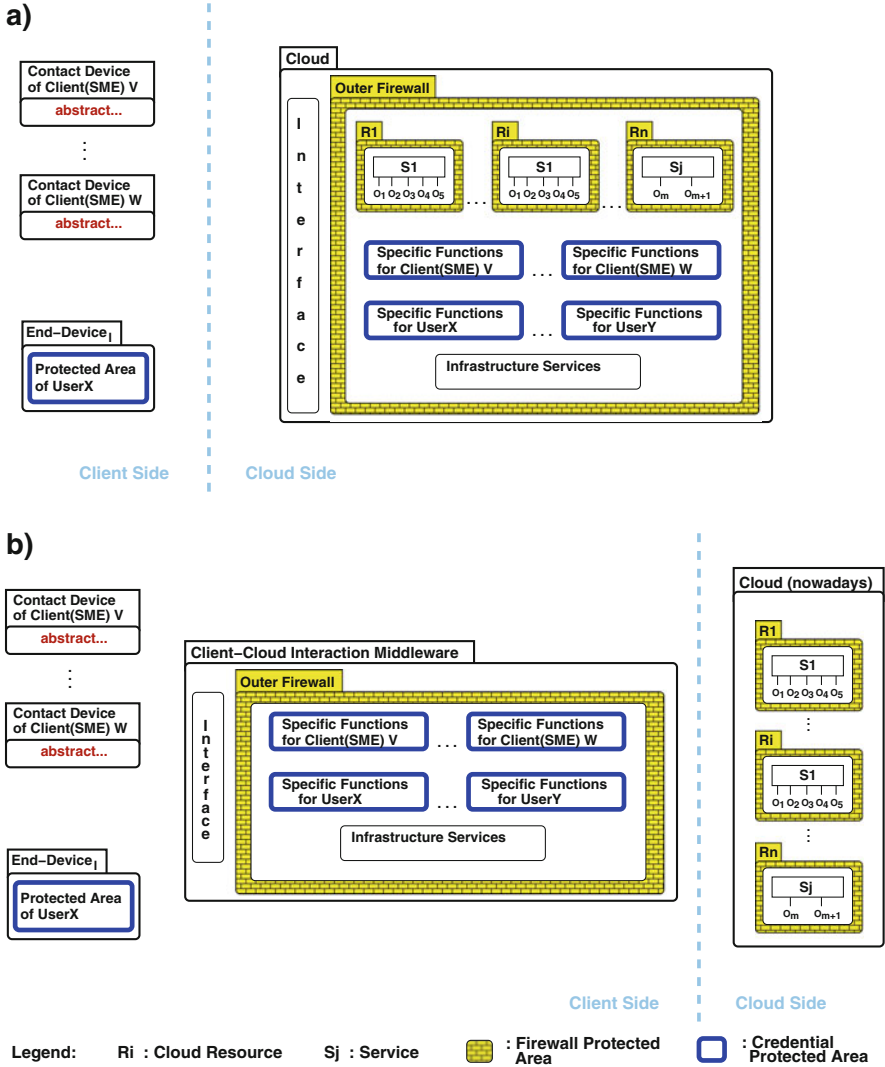


Fig. 2 Application of our model according to different scenarios. (a) Scenario I. (b) Scenario II

of and to provide a transparent and uniform access to more than one cloud. It should be noticed that the end users cannot bypass and omit the middleware and access to the cloud services, since they do not have direct access to the cloud(s) (they have credentials only to the middleware, and the middleware assigns permanently or temporarily their credentials to cloud service accounts).

In the following, we give an overview of various components of our integrated cloud service architecture, whose formal specifications are discussed in this chapter.

3.1 A Cloud Architecture Equipped with Service Plots and a Client-to-Client Interaction Feature

There are two major problems which we address with the discussed cloud architecture model [30]: on the one hand, how to provide a transparent and uniform way to interact with the end users, such that it allows the users to access and combine the available functions of cloud services which may belong to various cloud service owners (or tenants), and on the other hand, how to give full control into the hands of the tenants over the usage of the cloud resources which they own or rent on the cloud.

In our model, we assume that service instances are always equipped with *service operations* denoted by unique identifiers o_1, \dots, o_n ; see Fig. 2a. These service operations actually compose the interface of a service instance, and they are exported from a service to be used by other systems or directly by users.

Furthermore, it is assumed in our approach that each service owner has a dedicated contact point that resides out of the cloud. It is a special kind of client-side device that can also act as a server for the cloud itself in some cases. Namely, if a registered cloud user intends to subscribe to a particular service, he or she sends a subscription request to the cloud, which may forward it to such a special kind of client belonging to the corresponding service owner. This client responds with a special kind of action scheme called *service plot*, which algebraically defines and may constrain how the service can be used by the user (e.g., it determines the permitted combination of service operations). This special kind of client of the service owners is abstract in the current model.

The received service plots, which may be composed individually for each subscribing user by service owners, are collected with other cloud functions available for this particular user in a kind of personal user area by the cloud; see Fig. 2a. Later, when the subscribed user sends a service request, it is checked whether the requested service operations are allowed by any service plot. If a requested operation is permitted, then it is triggered to perform; otherwise, it is blocked as long as a plot may allow to trigger it in the future. Each triggered operation request is authorized to enter into the user area of the corresponding service owner to whom the requested service operation belongs. Here, a scheduler mechanism assigns to the request a one-off access to a cloud resource on which an instance of the corresponding service runs. Then the service operation request is forwarded to this resource, where the request is processed by an instance of the service whose operation was requested. Finally, the outcome of the performed operation returns to the area of the initiator user, where the outcome is either stored or sent further to a given end-user device.

In this way, the service owners have direct influence on the service usage of particular users via the provided service plots. If a user subscribes to more than one service, he or she may have access to more than one plot. These plots are independent from each other and they can be applied concurrently. If a service owner makes available more than one service for a user, the owner has the choice either to provide independent plots for the user or to combine some functions of various

services into a common service plot. This conceptual solution shows a transparent and uniform way how to provide an advanced access control mechanism for cloud services without giving up the flexibility of heterogeneous cloud access to these services.

Regarding our proposed cloud service model, one of the major questions can be whether it is adaptable to nowadays leading cloud architectures and solutions (e.g., Amazon S3, Microsoft Azure, IBM SmartCloud, etc.), since these systems rule the market at present and they will have impact on the cloud business in the near future as well. Since due to the applied ambient concept the relocation of the system component is trivial, we can apply our model according to different scenarios. For instance, all our novel methods including our client-cloud interaction solution can be shifted to the client side and wrapped into a middleware software which takes place between the end users and cloud in order to control their interactions; see Fig. 2b (this scenario was implemented in the mentioned software prototype). Note that the specified communication topology among the distributed system components remains the same in both proposed scenarios.

Thinking further this second scenario, we can envisage a new (cloud) service whose customers are enterprises that take over the role of the service owners in this service, such that they can fully control how their employees can access and use third-party clouds. Namely, assuming that these enterprises own or pay for various cloud services, they can provide customized service plots for their employees via such a client-cloud interaction controller service to restrict end-user accesses to the available functions of these cloud services.

3.1.1 Client-to-Client Interaction

We also extended the model of our cloud service architecture with a *client-to-client interaction (CTCI)* mechanism via a cloud architecture [29]. Our envisioned cloud feature can be regarded as a special kind of services we call *channels*, via which registered cloud users can interact with each other in an almost direct way and, what is more, they are able to share available cloud resources among each other as well.

Some use cases, which may claim the need of such CTCI functions, can be, for instance, disseminating large or frequently updated data whose direct transmitting meets some limitations or connecting devices of the same user (in the latter case, an additional challenge can be during a particular interpretation of the modeled CTCI functions, how to wrap and transport local area protocols, like *upnp* via the cloud). See an overview of the formal model of CTCI in Sect. 5.3.

3.2 *Client-Centric Identity and Access Management in Cloud Computing*

The adoption of cloud-based services offers many advantages for small and medium enterprises. However, a cloud-based approach also entails certain disadvantages. The papers [7, 34, 46, 99] outline some of these disadvantages: loss of control, contracting issues, provider lock-in, and other security and privacy issues. Such issues imply an extra level of trust between a client and a cloud provider. With respect to identity management, a loss of control implies that the client must trust the cloud provider with sometimes critical or important identity information such as credit card information. A potential client would prefer such data to be stored on premise and only be offered to a service on demand. The vendor lock-in issue might be mitigated by the adoption of services across multiple providers. In this scenario, a problem arises in maintaining identity data across the providers. Changing the surname, for example, entails changing this property for each service in particular (especially if the service provider does not adopt open standards). Our research is focused on providing a client-centric identity meta-system which allows a client to maintain a private identity directory while offering individual users automatic authentication to cloud-based services via a single sign-on, privacy-enhanced service.

In the paper [115], we have introduced the concept of an identity management machine (IdMM), an ASM-based, client-centric, single sign-on tool for small and medium enterprises that want to adopt or migrate to cloud-based services. This concept has been further refined in the paper [116] where we described the architecture of the IdMM. As mentioned in the paper [116], the IdMM is composed of six agents: the core agent (comprising the rules described in the paper [115]), the client agent (managing the interaction with the client's directory), the cloud agent (used for the interaction with a cloud service), the user agent (handling the interaction with an individual user), the protocol agent (used for protocol-based authentication), and the provisioning agent (managing user provisioning, password resets, and user de-provisioning). Apart from the core agent, each agent is defined by further refinement of the abstract functions presented in the paper [115]. This process is still an ongoing task, with the provisioning and protocol agents still left at an abstract level. In the paper [117], we have described the IdMM_{Client} agent by further refinement of the client-side abstract functions. We also gave an example of the IdMM_{Client} 's interaction with an ApacheDS LDAP directory [11].

The IdMM makes an abstraction of the protocols used by both the client and cloud provider for their identity management systems via the use of the abstract functions presented in the paper [115]. Such functions leave the organizational and implementation aspects of the identity management systems directly into the hands of the end parties. To describe these functions, we must first consider the interaction between a client and a cloud provider with respect to identity management, authentication, and authorization. We consider three distinct cases of

client-to-cloud interaction: the direct case, the obfuscated case, and the protocol-based case.

3.2.1 Direct Client-to-Cloud Interaction

As showed by the authors of the papers [7, 34, 50], one of the greatest issues surrounding identity management for cloud providers is the need of the cloud provider to control the customer experience. Many providers make use of their own custom-designed identity systems to which a client must subscribe. This means that the client has no choice but to use the cloud provider identity system. While this may be an inconvenience from a privacy point of view, the real problem lies in managing the client's information across multiple providers. A simple change, such as changing a user's address, entails changing the value on every single provider the client uses. Any change made on the client side must also be made on the cloud via the synchronization of attributes. Concurrently, any changes made by the provider (such as the addition, replacement, or removal of an attribute) must be reflected in the client's directory system.

3.2.2 Obfuscated Client-to-Cloud Interaction

While the direct client-to-cloud interaction allows for an efficient use of cloud services, it does suffer from a lack of privacy. Since all information about a client is stored on the provider's infrastructure, there is an increased risk that through data leakage or unauthorized access, that information could fall into the wrong hands. We mitigate this threat by introducing the concept of obfuscated identities.²

We consider an identity to be real if information contained corresponds to the identity's owner and is visible to any external entity. An obfuscated identity has its information obfuscated. Depending on the method of obfuscation, the information is either undecipherable or can only be deciphered by the owner of the identity. We also consider a third kind of identity, a *partially obfuscated identity*. Such an identity contains a mixture of real and obfuscated data. An example for the use of real identities can be found in the usage of online stores where the information must be accurate in order to process the payment and shipping. By contrast, one could use obfuscated identities for a free online storage service. If, for example, the storage service requires an age restriction, then one could use a partially obfuscated identity where only the date of birth is real.

The usage of obfuscated and partially obfuscated identities is dependent on the cloud service. As mentioned, some services do require real identities. Even if the service can be used with obfuscated identities, we leave the matter in the hands of the client. The client can choose whether or not to use obfuscation in such cases.

²For the purpose of this paper, we consider the definition of an identity as explained in paper [112].

For partially obfuscated identities, we also allow the client to choose what real data attributes are sent to a service.

3.2.3 Protocol-Based Client-to-Cloud Interaction

In recent years, there has been a drive to improve interoperability between cloud providers mostly to prevent vendor lock-in. From an identity management perspective, the result has been the adoption of some open-based protocols to facilitate both cloud interoperability as well as identity access management. Protocols such as OAuth or OpenID represent an important tool for a client-centric identity management system. They allow the provider to focus on the requirements of the service while allowing access via the standard implementation of these protocols. From the client's perspective, such protocols allow for an easier integration across multiple cloud providers. It must be noted however that such protocols do suffer from a variety of security and privacy issues, as described in [50, 88].

3.3 Cloud Content Adaptivity

This subsection is giving an overview of the problem of adaptivity to different services, devices, preferences, and environments. By adaptivity we understand that all the services provided by the cloud to the client should be adapted on the fly to the different contexts mentioned above.

The previously specified problem (initially presented in the paper [43]) is described in Fig. 3. As an example, we use a database manager application, which is deployed on the cloud. The user should be able to use the application (we can

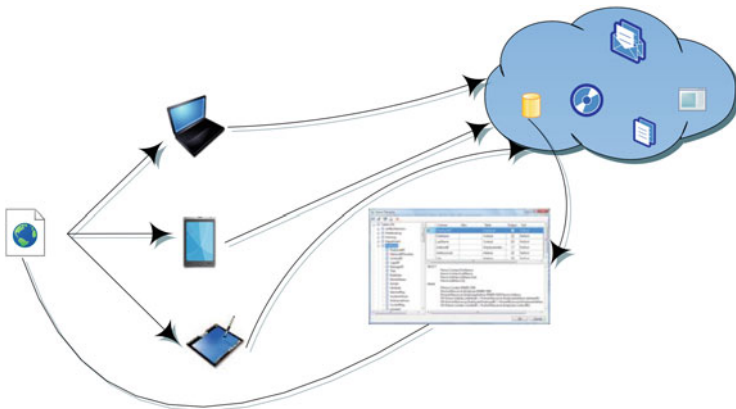


Fig. 3 Problem description

think of an application as of a list of services available in the cloud for that user) from any device he or she logs in without having to install any other application. How can we make the cloud services available on all devices? We need a general application that will adapt to different end devices on the fly. The implementation of a *Web application (WA)* comes as a solution to the client-cloud interaction, because it transfers data among different software components, which execute on different devices, using only the browser [85]. However, WAs do not have a precise definition or a precise model to follow, because they are related to various standards and implementation frameworks. To tackle this problem, we propose the conception of a formal model prior to code development that would include a rigorous analysis. Using formal modeling and verification, we want to guarantee reliability properties in order to ensure that, e.g., the client will receive the same (or as similar as possible) output independently of the device he or she is using. We show how requirements can be captured with ASM ground models and how the refinement method can be applied in order to link the ground models to pseudo-code. Together with the refinement method, the ASM ground models are generating a documentation which can be used for inspection, reuse, and maintenance.

We decided to use ASMs, instead of other modeling methods (e.g., *Unified Modeling Language (UML)*, *finite-state machine (FSM)*), for realizing the design of the client-cloud interaction system, because with them, we can prove that the system's development is correct and reliable (we can check if the WA under development behaves as expected). ASM method [25]:

- offers the possibility to design and analyze both procedural single-agent and asynchronous multiple-agent distributed systems (as the cloud framework we deal with). In ASMs, an action can be replaced in a refinement step by multiple parallel actions [22], which means that by going from the current state to the next state, the set of rules are executed simultaneously [23]. Because of these attributes, the ASM method was chosen over the FSM.
- creates a high-level modeling at the level of abstraction (allowing to describe complex systems, which can be easily understood by all the stakeholders) and links the descriptions in a chain of coherent system models (that can possibly bring to the implementation) using stepwise refinement. The former characteristic improves upon the loose character of UML description, and the second one also fills in a breach in UML [25].
- supports rigorous model validation and verification.

3.4 Cloudification Case Studies

In order to show and prove the feasibility and viability of our approach for controlling client-cloud interaction and managing access control, we have accomplished two cloudification case studies.

In the first case study, we took a stand-alone application called visual SQL [45, 66, 111], whose source code was available for us. Visual SQL is a software tool which provides a graphical database description language to facilitate intelligent conceptual diagramming of database queries. We adapted this software to an existing cloud architecture and made it interoperable with the prototype of our *client-cloud interaction middleware* software such that it provides access control for this cloudified application. This work shows how software applications which were originally designed for single usage on a local desktop use can be adapted to a cloud infrastructure and equipped with a sophisticated access control mechanism for multiple users.

The second case study was related to Office 365 which is a subscription-based cloud service provided by Microsoft and which integrates other Microsoft online services together such as Exchange Online, SharePoint Online, Lync Online, Web Apps, and Office Professional Plus. We combined Office 365 with our client-cloud interaction middleware such that its functionality was extended with an access control mechanism, which is more sophisticated than its own. Namely, Office 365 provides only a limited authority for customers/admin to control their users.

This advanced access control which is achieved by the application of the client-cloud interaction middleware cannot be bypassed by the end users, since they have direct access only to the middleware, but not to the purchased Office 365 service package of any customer (company) of client-cloud interaction middleware. For instance, customers of client-cloud interaction middleware can restrict which end user can make a copy from e-mails received in his or her Office 365 mailbox, who can attach document stored in Office 365 to e-mails, or who can send e-mails to external e-mail addresses using Office 365 mail service. We should emphasize that this work was carried out in the way that we discovered and used only the publicly available programming interfaces and tools for Office 365 and we did not get any extra support from Microsoft.

An implemented prototype in the cases of both case studies, respectively, was deployed for testing purposes on Windows virtual machines under OpenStack cloud infrastructure software on an IBM CloudBurst server machine as well. It is beyond the scope of this chapter to describe these case studies and their formal specification, but we refer to [103], which discusses the first case study and its formal model in details.

4 Preliminaries

In this section, we give a short summary on the applied formal approach as well as ambient calculus and ambient ASM in order to facilitate the understanding of the latter sections.

4.1 The Applied Formal Approach

For our research, we searched for a software engineering method by which both the algorithms of the concurrent system components and the dynamic topology of complex distributed and cloud systems can be formally described. As a first step, we investigated the following two formal approaches:

- One of the most outstanding methods for formal modeling of distributed components of (mobile) network applications is a calculus of mobile agents called *ambient calculus* [38, 61]. This concept is simple, succinct, and sufficient enough to efficiently describe locality as well as phenomena related to mobility.

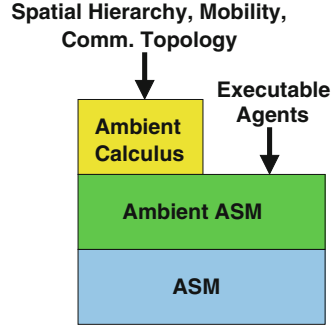
Additionally, besides mobility, ambient calculus inherently supports the reasoning of high-level abstraction of many security considerations which can be defined by formulating the ability or inability of various entities to cross certain physical or virtual barriers [37] (e.g., certain combinations of some ambient expressions can be interpreted as firewalls that filter traffic according to some security policies, others can be regarded as abstractions of certain access controls or of ciphertext decodings, etc.). But one of its main drawbacks is that the ambient calculus is not capable to treat the algorithmic specification of the executable agents which appear in its ambient constructs.

- For this latter purpose, an obvious candidate would be the mathematically well-funded and efficient software engineering method called *abstract state machines* (ASMs) [25, 64]. ASMs have already demonstrated their usefulness and their viability in many fields, for example, in the formal definition of sequential [65] and parallel algorithms [19, 20], giving comprehensive high-level definition and analysis (verification and validation) of Java programming language and of the Java virtual machine [108], and modeling Web services [9]. However, there is a limitation in ASMs to describe dynamically changing hierarchical structures of components in distributed systems and to express some system properties which depends on their distribution in space (e.g., local deadlock).

In [26], the ambient concept (notion of “nestable” environments where computation can happen) is introduced into the ASM method. In that article, it is also shown how to encode the mobile ambients of ambient calculus in terms of *ambient ASM* rules. Since one of the main goals of [26] is to reveal the inherent opportunities of the new ambient concept introduced into ASMs, the presented definitions for moving ambients are unfortunately incomplete.

In [31], we extended and completed the ASM rules mentioned above, such that they fully capture the ambient calculus. By this, a new method is created in terms of ASM rules, in which one is able to describe formal models of distributed systems including mobile components in two different abstraction layers (see Fig. 4). This means that while the algorithms and local interactions of executable agents are given in terms of ASMs, the long-term interactions and movements of system components via various administrative domains are specified with the terms of ambient calculus

Fig. 4 Abstraction layers in the chosen formal approach



in our approach. This novel method makes possible in a given model that:

- the agents may not only be arbitrarily placed and linked initially, but the composed structure is rearranged from time to time by the programs of agents residing in it;
- the behavior of agents is influenced by their current spatial locations and communication topology;
- one can express visibility and access conditions on ASM agents (or on some administrative boundaries) explicitly.

Since the definition of ambient ASM is based upon the semantics of ASM without any changes, each specification given this way can be translated into a traditional ASM specification.

4.2 Ambient Calculus

The ambient calculus [38] was inspired by the π -calculus [82], but it focuses primarily on the concept of locality and process mobility across well-defined boundaries instead of channel mobility as π -calculus. The concept of *ambient* stands in the center of the calculus; see a summary of the definition of ambient calculus in Table 1.

The ambient calculus includes only the mobility and communication primitives depicted in Table 1(A).³ The main syntactic categories are *processes* (including both ambients and agents) and *actions* (including both *capabilities* and *communication primitives*). A reduction relation $P \longrightarrow Q$ describes the evolution of a term P into a new term Q (and $P \longrightarrow^* Q$ denotes a reflexive and transitive reduction relation from P to Q).

³Name restriction creates a new (unique) name n within a scope P . One must be careful with the term $!(\nu n)P$, because it provides a fresh value for each replica, so $!(\nu n)!P \neq !(\nu n)P$.

Table 1 Definition of ambient calculus

(A) The mobility and communication primitives		(B) Reduction (operational semantics)
$P, Q, R ::=$	Processes	$P \equiv P', Q \equiv Q', P \rightarrow Q \implies P' \rightarrow Q'$
$P \mid Q$	Parallel composition	
$n[P]$	Ambient	$P \rightarrow Q \implies P \mid R \rightarrow Q \mid R$
$(\nu n)P$	Restriction of name n within P ³	
0	Inactivity (skip process)	$P \rightarrow Q \implies n[P] \rightarrow n[Q]$
$!P$	Replication of P	
$M.P$	(Capability) action M then P	$P \rightarrow Q \implies (\nu n)P \rightarrow (\nu n)Q$
$(x).P$	Input action (the input value is bound to x in P)	$n[\text{IN } m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$
$\langle a \rangle$	Async output action	$m[n[\text{OUT } m.p \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$
$M_1. \dots .M_k$	A path formation on actions	
$M ::=$	Capabilities	$\text{OPEN } n.P \mid n[Q] \rightarrow P \mid Q$
$\text{IN } n$	Entry capability (to enter n)	
$\text{OUT } n$	Exit capability (to exit n)	$(x).P \mid \langle a \rangle \rightarrow P(x/a)$
$\text{OPEN } n$	Open capability (To dissolve n 's boundary)	
(C) Structural congruence (operational semantics)		
$P \equiv P$		$P \equiv Q \implies Q \equiv P$
$P \equiv Q, Q \equiv R \implies P \equiv R$		$P \equiv Q \implies P \mid R \equiv Q \mid R$
$P \equiv Q \implies n[P] \equiv n[Q]$		$P \equiv Q \implies !P \equiv !Q$
$P \equiv Q \implies (\nu n)P \equiv (\nu n)Q$		$P \equiv Q \implies M.P \equiv M.Q$
$P \equiv Q \implies (x).P \equiv (x).Q$		$P \mid Q \equiv Q \mid P$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$		$!P \equiv P \mid !P$
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$		$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin \text{fn}(P)$
$(\nu n)(m[P]) \equiv m[(\nu n)P] \quad \text{if } n \neq m$		$P \mid 0 \equiv P$
$!0 \equiv 0$		$(\nu n)0 \equiv 0$

An ambient is defined as a bounded place where computation happens. An ambient is written as $n[P]$, where n is its name, which can be used to control access (entry, exit, communication, etc.), and a process P is running inside its *body* (P may be running even if n is moving). Ambient names may not be unique. Ambients can be embedded into each other such that they can form a hierarchical tree structure. An ambient body is interpreted as the parallel composition of its elements (its local ambients and its local agents) and can be written as follows:

$$n[P_1 \mid \dots \mid P_k \mid m_1[\dots] \mid \dots \mid m_l[\dots]] \text{ where } P_i \neq m_i[\dots]$$

An ambient can be moved. When an ambient moves, everything inside it moves with it (the boundary around an ambient determines what should move together with it). An action defined in the calculus can precede a process P . P cannot

start to execute until the preceding actions are performed. Those actions that are able to control the movements of ambients in the hierarchy or to dissolve ambient boundaries are restricted by capabilities. By using capabilities, an ambient can allow some processes to perform certain operations without publishing its true name to them (see the entry, exit, and open in Table 1). In case of the modeling of a real-life system, communication of (ambient) names should be rather rare, since knowing the name of an ambient gives a lot of control over it. Instead, it should be common to exchange restricted capabilities to control interactions between ambients (from a capability, the ambient name cannot be retrieved).

4.3 Ambient ASM

The core idea of ambient ASM [26] is to introduce an implicit parameter *curamb* to each location expressing a context for evaluation of terms and execution of machines. Analogously to conventional implicit object-oriented parametrization (e.g., *this.f(x) = f(x)*), the *dot term s.t* is introduced, where *s* is a term standing for an ambient expression, *t* is a term of the form $f(t_1, \dots, t_n)$ and *f* is a *location symbol*.

To each location, an additional argument is added for the ambient *curamb* in which the location is evaluated. Moreover, the basic ASM classification of functions and locations is extended with *ambient-independent functions and locations* (i.e., static or dynamic functions and location) whose values for a given argument do not depend on any ambient expression.

An ASM is an ambient ASM if it can be obtained from a basic ASM [25] by allowing for every given machine *P* also a machine of the following form:

$$\mathbf{amb\ } exp\ \mathbf{in\ } P$$

where execution of *P* is performed in the ambient *exp* (*exp* is bound to *curamb*). Additionally, the notation $n[P]$ introduced by Cardelli and Gordon [38] is also defined in ambient ASMs as follows:

$$n[P] = \mathbf{amb\ } n\ \mathbf{in\ } P$$

The semantics of ambient ASMs is defined by transformation into basic ASMs in [26].

4.3.1 Moving Ambients

In [26], an ASM machine called MOBILEAGENTSMANAGER is described as well, which gives a natural formulation for the reduction of the three basic capabilities (ENTRY, EXIT, and OPEN) of ambient calculus in terms of ambient ASM rules. For

this machine, an ambient tree hierarchy is always specified initially in a dynamic-derived function called *curAmbProc*. The machine `MOBILEAGENTSMANAGER` transforms the current value of *curAmbProc* according to the capability actions given in *curAmbProc*.

In [31], we presented an extended version of the ASM machine `MOBILEAGENTS-MANAGER`, which fully captures the calculus of mobile agents⁴ and which is also able to interpret (in the corresponding ambient contexts) the agents' algorithms given in terms of ASM syntax in *curAmbProc*. This updated ASM machine is called `EXTENDEDMOBILEAMBIENTMANAGER`.

One of consequences of this is that one is able to describe formal models of distributed systems including mobile components in the mentioned two abstraction layers and apply some experimental validation on these models. This means that:

- a particular system model can be defined as part of the value of *curAmbProc* in terms of ambient calculus terms (like what we did in Sect. 5.2) and in terms of ASMs;
- either models of some external instruments or some (user) actions (see in Sect. 5.1) can additionally be added to *curAmbProc* in parallel composition with the given model (they will act the role of the environment of the system in a particular case).

The value of *curAmbProc* will serve as input for the ASM machine `EXTENDEDMOBILEAMBIENTMANAGER`, which will transform the given value of *curAmbProc* (by performing the possible reduction steps and by parsing the specified ASMs) in order to check how the defined model interacts with its environment and how it behaves in a particular situation.

4.4 Definitions

As Cardelli and Gordon showed in [38], the ambient calculus with the three basic capabilities (`ENTRY`, `EXIT`, and `OPEN`) is powerful enough to be Turing complete. But for facilitating the specification of such a compound formal model as a model of a cloud infrastructure, we defined some new *nonbasic capability* actions encoded in terms of the three basic capabilities.

All the new ambient calculus capabilities defined in this section either directly appear in the presented cloud model in Sect. 5.2 or serve as a basis and were utilized for the definitions of one or more subsequent capabilities presented in this section as well. Table 2 summarizes the definitions of these nonbasic capabilities.

⁴Besides the three basic capabilities, the reductions of name restriction, input action, and asynchronous output action as well as the structural congruence for replication are also defined in terms of ambient ASM rules.

Table 2 A summary of the definitions of some nonbasic capabilities

Names	New reduction relations (based on the definitions)	Definitions of the new capabilities
1. Renaming	$n \mid n \text{ BE } m.P \mid Q \longrightarrow^* m \mid P \mid Q \mid$	$n \text{ BE } m.P \equiv (v \ s)(s \mid \text{OUT } n \mid m \mid \text{OPEN } n.\text{OUT } s.P \mid \mid \text{IN } s.\text{IN } m)$
2. Seeing	$n \mid \mid \text{SEE } n.P \longrightarrow^* n \mid \mid P$	$\text{SEE } n.P \equiv (v \ r, s)(r \mid \text{IN } n.\text{OUT } n.r \text{ BE } s.P \mid \mid \text{OPEN } s)$
3. Wrapping	$n \mid m \text{ WRAP } n.P \longrightarrow^* m \mid n \mid P \mid \mid$	$m \text{ WRAP } n.P \equiv (v \ s, r)(s \mid \text{OUT } n.\text{SEE } n.s \text{ BE } m.r \mid \text{IN } n \mid \mid \mid \text{IN } s.\text{OPEN } r.P)$
4. Allowing code	$\text{ALLOW } \text{Key}.P \mid \text{Key} \mid Q \longrightarrow^* P \mid Q$	$\text{ALLOW } \text{Key}.P \equiv \text{OPEN } \text{Key}.P$
5. Drawing in (an Ambient)	$m \mid Q \mid \text{ALLOW } \text{Key} \mid \mid n \mid n \text{ DRAWIN}_{\text{key}} m.P \mid \longrightarrow^* n \mid Q \mid P \mid$	$n \text{ DRAWIN}_{\text{key}} m.P \equiv \text{Key} \mid \text{OUT } n.\text{IN } m.\text{IN } n \mid \mid \text{ALLOW } m.P$
6. Drawing in Then Release a Lock	$m \mid Q \mid \text{ALLOW } \text{Key} \mid \mid n \mid \text{DRAWIN}_{\text{key}} m \text{ THEN-RELEASE } \text{lock}.P \longrightarrow^* \text{lock} \mid n \mid Q \mid P \mid \mid$	$n \text{ DRAWIN}_{\text{key}} m \text{ THENRELEASE } \text{lock}.P \equiv \text{Key} \mid \text{OUT } n.\text{IN } m.\text{IN } n \mid \mid \text{SEE } m.\text{lock } \text{WRAP } n.\text{ALLOW } m.P$
7. Concurrent Server Process	$m \mid Q \mid \text{ALLOW } \text{Key} \mid \mid \text{SERVER}_{\text{key}}^n m.P \longrightarrow^* \text{SERVER}_{\text{key}}^n m.P \mid n_k^{m \mid Q} \mid Q \mid P \mid$	$\text{SERVER}_{\text{key}}^n m.P \equiv (v \ \text{next})(\text{next} \mid \mid \mid \mid (v \ n)(\text{OPEN } \text{next}.n \mid n \text{ DRAWIN}_{\text{key}} m.P \text{ THENRELEASE } \text{next}.P))$

4.4.1 Applied Notations

In the rest of this chapter, the term $P \longrightarrow^* Q$ denotes multiple reductions. In addition, $P \xrightarrow{asm}^* Q$ denotes one or more steps of some ASM agents. We also apply the following abbreviations:

$$\begin{aligned} M_1. \dots M_n &\equiv M_1. \dots .M_n.0 \text{ where } 0 = \text{inactivity} \\ n[] &\equiv n[0] \text{ where } 0 = \text{inactivity} \\ (v n_1, \dots, n_m)P &\equiv (v n_1) \dots (v n_m)P \end{aligned}$$

4.4.2 Nonbasic Capabilities

Below we give an informal description of each nonbasic capability in Table 2. It is beyond the scope of the chapter to present detailed explanations and reductions of their ambient calculus-based definitions, but we refer to our former works [30] and [28] for more details.

1. **Renaming** This capability is applied to rename an ambient comprising this capability. Such a capability was already given in [38], but our definition differs from Cardelli's definition. In the original definition, the ambient m was not enclosed into another, name-restricted ambient (it is called s in our definition), so after it has left ambient n , n may enter into another ambient called m (if more than one m exist as sibling of n).
2. **Seeing** This operation was defined in [38], and it is used to detect the presence of a given ambient.
3. **Wrapping** Its aim is to pack an ambient comprising this capability into another ambient.
4. **Allowing Code** This capability is just a basic OPEN capability action. It is applied if an ambient allows/accepts an ambient construct (which may be a bunch of foreign codes) contained by the body of one of its sub-ambients (which may be sent from a foreign location). The name of the sub-ambient can be applied for identifying its content, since its name may be known only by some trusted parties.
5. **Draw in (an Ambient)** The aim of this capability is to draw in a particular ambient (identified by its name) into another ambient (which contains this capability) and then to dissolve this captured ambient in order to access its content. For achieving this, a mechanism (contained by the ambient key) is applied which can be regarded as an abstraction of a kind of protocol identified by key . The ambient key enters into one of the available target ambients which should accept its content in order to be led into the initiator ambient.
6. **Draw in then Release a Lock** This capability is very similar to the previous one, but after m has been captured by n (and before m is dissolved), n is wrapped by

another ambient. The new outer ambient is usually employed as release for a lock.⁵

- 7. Concurrent Server Process** This ambient construct can be regarded as an abstraction of a multi-threaded server process. It is able to capture and process several ambients having the same name in parallel. In the definition, n is a replicated ambient whose each replica is going to capture another ambient called m . Since there is a name restriction quantifier in the scope of the replication sign, which bounds the name n , a new, fresh, and unique name (denoted by n_k^{uniq}) is generated for each replica of n . One of the consequences of this is that nobody knows from outside the true name of a replica of the ambient n , so each replica of n is inaccessible from outside for anybody (even for another replica of n too).

5 The Specification of the Cloud Service Architecture Based on Ambient ASM

As was explained at the end of Sect. 4.3.1, the model of our cloud architecture is given as part of the value of the dynamic-derived function called *curAmbProc*, which serves as input for the ASM machine EXTENDEDMOBILEAMBIENTMANAGER [31].

$$curAmbProc := root[Cloud | Client_1 | \dots | Client_n]^6$$

In this section, we focus on the system configuration scenario, where the model of our new software solutions is integrated with a cloud model (see Fig. 2a); however, due to the applied ambient concept, the relocation of the new infrastructure services into a middleware component within the model (see Fig. 2b) cannot be a problem.

In the formal model discussed in this section, we assume that there are some standardized public ambient names, which are known by all contributors. We distinguish the following kinds of public names: addresses (e.g., *cloud*, *client*₁, ..., *client* _{n}), message types (e.g., *reg(istration)*, *request*, *subs(cription)*, *returnValue*, etc.), and parts of some common protocols (e.g., *lock*, *msg*, *intf*, *access*, *out*, *o*₁, ..., *o* _{s} , *op*). All other ambient names are nonpublic in the model which follows.

In this section, we leave the end devices on the client-side abstract, but we define some user actions with which the cloud service architecture is able to interact.

⁵In ambient calculus, the capability $OPEN\ n.P$ is usually used to encode locks [38]. Such a lock can be released with an ambient like $n[Q]$ whose name corresponds with the target ambient of the $OPEN$ capability.

⁶The ambient called *root* is a special ambient which is required for the ASM definition of ambient calculus; see [26] and [31].

5.1 User Actions

In the model, user actions are encoded as messages. A user can send the following kinds of messages to the cloud:

$MsgFrame \equiv msg[\text{IN } cloud.AALLOW \text{ } intf.content]$
where $content$ can be
 $RegMsg \equiv reg[\text{ALLOW } CID.\langle UID_x \rangle]$
 $SubsMsg \equiv subs[\text{ALLOW } CID.\langle UID_x, SID_i, pymt \rangle]$
 $RequestMsg \equiv request[\text{IN } UID_x |$
 $o_i[\text{ALLOW } op.\langle client_k, args_i \rangle] |$
 \vdots
 $o_j[\text{ALLOW } op.\langle client_k, args_j \rangle]]$
 $AddClMsg \equiv addCl[\text{IN } UID_x | \text{ALLOW } CID.\langle client_k, path_l, UID_{(on\ client_l)} \rangle]$
 $AddChMsg \equiv addCh[\text{ALLOW } CID.\langle UID_x, cname \rangle]$
 $SubsToChMsg \equiv subsToCh[\text{ALLOW } CID.\langle UID_x, cname, uname, client_k, pymt \rangle]$
 $ShareInfoMsg \equiv share[\text{IN } CHID_i | \text{ALLOW } CID.\langle sndr, rcvr, info \rangle]$
 $ShareSvcMsg \equiv share[\text{IN } CHID_i | \text{ALLOW } CID.\langle sndr, rcvr, info, o_i, argsP, argsF \rangle]$

In the definitions above, the ambient msg is the frame of a message; the term $\text{IN } cloud$ denotes the address to where the message is sent; the term $\text{ALLOW } intf$ allows a (server) mechanism on the target side which uses the public protocol $intf$ to capture the message; and the $content$ can be various kinds of message types. The term $\text{ALLOW } CID$ denotes that the messages are sent to a service of a particular cloud which identifies itself with the nonpublic protocol/credential CID (stands for *cloud identifier*).

The first three kinds of messages were introduced in the original model. In a *RegistrationMsg*, the user x provides his or her identifier UID_x that he or she is going to use in the cloud. By a *SubscriptionMsg*, a user subscribes to a cloud service identified by SID_i ; the information represented by $pymt$ proves that the given user has paid for the service properly.

Again, cloud services provide their functionalities for their environment (users or other services) via actions called service operations in our model. In a *RequestMsg*, a user who has subscribed to some services before can request the cloud to perform some service operations belonging to some of these services. o_i and o_j are the unique names of these service operations and denote service operation requests; $client_k$ is the identifier of a target location (usually a client device) to where the output of a given operation should be sent by the cloud; and $args_i$ and $args_j$ are the arguments of the corresponding requested service operations. Furthermore, the term $\text{IN } UID_x$ represents the address of the target user area within the *cloud*, and $\text{ALLOW } op$ denotes that the request will be processed by a service plot, which expects service operation requests (and which interacts with the request via the public protocol op).

The rest of the message types is used by the CTCI functions; see Sect. 5.3. With *AddCIMsg*, a user can register a new possible target (client) device or location for the outcomes of the requests initiated by him or her. Such a message should contain the chosen identifier $client_k$ of the new device, the address $path_l$ of the device, and the user identifier $UID_{(on\ client_l)}$ used on the given target device.

By *AddChMsg*, users can open new channels; by *SubsToChMsg*, users can subscribe to channels; and by *ShareInfoMsg* and *ShareSvcMsg*, users can share information as well as service operations with some other users registered in the same channel. For the detailed description of the argument lists of these last four messages, see Sects. 5.3.1–5.3.3.

5.2 The Formal Model of the Cloud Architecture

The basic structure of the defined cloud model, which is based on the simplified *infrastructure as a service (IaaS)* specification given in [31], is the following:

```
cloud ≡ (v fw, q, rescr1, . . . rescrm)cloud[
  interface |
  fw [ rescr1[ service1 ] | . . . | rescrl[ service1 ] | rescrl+1[ service2 ] | . . . | rescrm[ servicen ] |
  q [ !OPEN msg | BasicCloudfunctions | CTCIfunctions |
  UIDx[ userIntf ] | . . . | UIDy[ userIntf ] |
  UIDvowner[ ownerIntf ] | . . . | UIDwowner[ ownerIntf ]
]]]
where
  interface ≡ SERVERinfn msg.IN fw.IN q.n BE msg
```

In the cloud definition above, the names of the ambients fw , q , and $rescr_1, \dots, rescr_m$ are bound by name restriction. The consequence of this is that the names of these ambients are known only within the cloud service system, and therefore the contents of their body are completely hidden and not accessible at all from outside of the cloud. So each of them can be regarded as an abstraction of a firewall protection.

The ambient expression represented by *interface* “pulls in” into the area protected by the ambients fw and q any ambient construct which is encompassed by the message frame msg . The purpose of the restricted ambients fw and q is to prevent any malicious content which may cut loose in the body of q after a message frame (msg) has been broken (by OPEN msg) to leave the cloud together with some sensitive information. For more details, we refer to [30].

The restricted ambients $rescr_1, \dots, rescr_m$ represent computational resources of the cloud. Within each cloud resource, some service instances can be deployed. A service may have several deployed instances in a cloud (see instances of $service_1$ in $rescr_1, \dots, rescr_l$ above).

Every user area is represented by an ambient whose name corresponds to the corresponding user identifier UID_i . Furthermore, the user areas extended with service

owner role are denoted by UID_i^{owner} . The terms denoted by *BasicCloudfunctions* are responsible for cloud user registration and service subscription. Finally, the terms denoted by *CTCIfunctions* encode the client-to-client interaction.

It is beyond the scope of this chapter to describe all parts of this model in detail (e.g., the structure of service instances $service_i$, the functions of a service owner area $ownerIntf$, and the ASM agents in *BasicCloudfunctions*). For the specification of these components, we refer to [30].

5.2.1 User Access Layers

A user access layer (or user area) may contain the following mechanisms: accepting user requests (!ALLOW *request*), accepting new plots (!ALLOW *newPlot*), and accepting outputs of service operations (!ALLOW *returnValue*) and some service plots.

$userIntf \equiv$
 !ALLOW *request* | !ALLOW *newPlot* | $clientRegServer$ | !ALLOW *returnValue* |
 PLOT $_{SID_i}$ | ... | PLOT $_{SID_j}$ |
 $sortingOutput$ | $client_1[posting_{client_1}]$ | ... | $client_k[posting_{client_k}]$
where
 $sortingOutput \equiv !(o, client, a).output[IN client.ALLOW CID | \langle o, client, a \rangle]]$
 $clientRegServer \equiv SERVER_{CID}^n addCl.(client, path, UID).(n BE client | posting_{client})$
 $posting_{client_i} \equiv SERVER_{CID}^n output.(o, client, a).$
 OUT $client_i.forwardTo_{client_i}.returnValue[IN UID_{(on client_i)} | \langle o, client, a \rangle]]$
 $forwardTo_{client_i} \equiv n BE outgoingMsg.OUT UID_x.leavingCloud.path_i$
 $leavingCloud \equiv OUT q.OUT fw.OUT cloud.outgoingMsg BE msg$

$clientRegServer$ is applied to process every *AddCIMsg* sent by the corresponding user. It creates new communication endpoint for target (client) devices. Each endpoint is encoded by an ambient whose name $client_i$ corresponds with the given identifier provided in a message *AddCIMsg*. By these endpoints, outputs of service operations can immediately be directed to registered (client) devices after they are available. Of course, if no target device or a non-registered one is given in a *RequestMsg*, the outcome will be stored in the area of the user.

Every service operation output, which is always delivered within the body of an ambient called *returnValue*, consists of three parts: the name of the performed service operation, the identifier of a target location to where the output should be sent back, and the outcome of the performed service operation itself.

$sortingOutput$ distributes every service operation output among the communication endpoints in an ambient called *output*. The mechanism $posting_{client_i}$, which resides in each communication endpoint, is responsible to wrap each output of service operations which reaches the corresponding endpoint again into an ambient *returnValue* and to forward it to the specified user $UID_{(on client_i)}$ on the corresponding device $client_i$. It is beyond the scope of this chapter to present a reduction on how a simple request is processed in our model, but we refer to [30] for more details.

5.2.2 Service Plots

According to [96], a plot is a high-level specification of an action scheme, i.e., it captures possible combination of actions (e.g., service operations) in order to perform a certain task. As it has been mentioned in Sect. 3.1, in our model, service plots are provided by the service owners who have disposal of access rights of the services and they are placed into the user areas during the service subscriptions. The purpose of their usage is to determine the permitted combination of service operations which can be used by the user with his/her valid subscriptions.

For an algebraic formalization of plots, *Kleene algebras with tests (KATs)* [70] have been applied in [94, 95]. Then a plot is an algebraic expression that is composed out of elements of a carrier-set K containing two elementary operations 0, 1 and elementary processes (or propositional atoms), of binary operators \cdot and $+$, and of unary operators $*$ and $\bar{}$, the last one being only applicable to propositions.

For our purposes, we employ a simplified definition of service plots where we leave the propositional ground of KATs. This means that in a service plot, the elementary processes (e.g., $o_i, o_j \in K$) correspond to *slots* for operation requests such that each slot can accept and permit a request only for a particular service operation, \cdot denotes a sequence (e.g., $o_i \cdot o_j$ or $o_i o_j$), $+$ denotes a choice (e.g., $o_i + o_j$), and $*$ denotes iteration (e.g., o_i^*). In addition, we can simulate parallelism by equating $o_i \mid o_j$ with $o_i o_j + o_j o_i$; so in fact, we are interested in interleaved, concurrent service operations when we talk of *parallel composition* of plots.

It is beyond the scope of the chapter to discuss service plots in detail; for more information, see [30].

5.3 Client-to-Client Interaction Feature

Again, the client-to-client interaction in our model is based on the constructs called *channels*. These are represented by ambients with unique names denoted by $CHID_i$ which contain some mechanisms whose purpose is to share some information and service operations among some subscribed users; see below:

CTCI functions \equiv
 $CHID_1[\text{channelIntf}] \mid \dots \mid CHID_l[\text{channelIntf}] \mid$
 $SERVER_{CID}^n \text{addCh.}(UID, cname).CHMGR(n, UID_x, cname) \mid$
 $SERVER_{CID}^n \text{subsToCh.}(UID, cname, uname, client, pymt).$
 $CHSUBSMGR(n, UID, cname, uname, client, pymt)$
where
 $\text{channelIntf} \equiv SERVER_{CID}^n \text{share.}$
 $((sndr, rcvr, info).(sndr, rcvr, info, \mathbf{undef}, \mathbf{undef}, \mathbf{undef}) \mid$
 $(sndr, rcvr, info, o, argsP, argsF).SHARINGMGR(n, sndr, rcvr, info, o, argsP, argsF))$

The specification above contains three ASM agents called CHMGR, CHMGR, and SHARINGMGR, whose complete ASM definitions are given in [29], but it is beyond the scope of this chapter to describe them here.

Every cloud user can create and own some channels by sending the message *AddChMsg*⁷ to the cloud, where an instance of the ASM agent CHMGR, which is equipped with a server mechanism, processes such a request and creates a new ambient with unique names for the requested channel; see Sect. 5.3.1.

If a user would like to subscribe to a channel, he or she should send the message *SubsToChMsg* to the cloud. The server construct belonging to the ASM agent CHSUBSMGR is responsible for processing these messages; see Sect. 5.3.2. In the subscription process, the owner of the channel can decide about the rights which can be assigned to a subscribed user. According to the presented high-level model, the employed access rights are encoded by the following static nullary functions: *listening* is a default basic right, because everybody who joins a channel can receive shared contents; *sending* authorizes a user to send something to only one user at a time; and *broadcasting* permits a user to distribute contents to all members of the channel at once.

Both *ShareInfoMsg* and *ShareSvcMsg* are processed by the same server which belongs to the ASM agent SHARINGMGR and which is located in the body of each ambient *CHID_i*; see Sect. 5.3.3. In the case of *ShareInfoMsg*, the server first supplements the argument list of the message with three additional **undef** values, such that it will have the same number of arguments as *ShareSvcMsg* has. Then an instance of the ASM agent SHARINGMGR can process the *ShareInfoMsg* similarly to *ShareSvcMsg* (the first three arguments are the same for both messages).

5.3.1 Establishing a New Channel

CHMGR is a parameterized ASM agent, which expects *UID* of the cloud user who is going to create a new channel and *cname* which is the name of this channel as arguments. The additional argument *n* is the unique name of an ambient which was provided by the surrounding server construct and in which the current *AddChMsg* is processed by an instance of this agent (such an argument is also applied in the case of the other ASM agents below).

First, the agent checks whether the given *UID* has already been registered on the cloud and whether the given name *cname* has not been used as a name of an existing channel yet. If it is the case, the agent generates a new and unique identifier denoted by *CHID* for the new channel; and it inserts into an abstract database a new entry with all the details of the new channel which are the channel identifier, the channel name, and the identifier of the owner.

Then it creates an ambient called *CHID* with the terms denoted by *channelIntf* in its body which encode the functions of the new channel. By the abstract tree

⁷User actions *AddChMsg*, *SubsToChMsg*, *ShareInfoMsg*, and *ShareSvcMsg* are defined in Sect. 5.

manipulation operation called `NEWAMBIENTCONSTRUCT`⁸ introduced in [31], this generated ambient construct is placed into the ambient tree hierarchy as sibling of the agent.

Although a channel is always created as a sibling of the current instance of `CHMGR`, it contains the capability action `OUT n`; so as a first step, it leaves the ambient n which was provided by the surrounding server construct and in which the message was processed. After that, it is prepared to serve as a channel for client-to-client interaction (it is supposed that the name *cname* of every channel is somehow announced among the potential users).

5.3.2 Subscribing to a Channel

`CHSUBSMGR` is a parameterized ASM agent, which expects the following arguments: *UID* of the user who is going to subscribe to the channel, *cname* which is the name of the channel, *uname* which is the name that the user is going to use within the channel, *client* which is the identifier of a registered client device to where the shared content will be forwarded, and *pymt* which is some payment details if it is required. A user can register to a channel with different names and various client devices in order to connect these devices via the cloud.

First, the agent checks whether the given *UID* and *cname* have already been registered on the cloud and whether the given *uname* has not been used as a name of a member of the channel yet. If it is the case, the agent informs the owner of the channel about the new subscription, who responses with a set of access rights to the channel that he or she composed based on the information given in the subscription.

If the subscription has been accepted by the owner and besides *listening*, some other rights are granted to the new user, an ambient construct is created and sent as a message *returnValue* to the user by `NEWAMBIENTCONSTRUCT`. This message contains the capability `IN CHID` by which the new user can send messages called *ShareInfoMsg* and *ShareSvcMsg* into the ambient *CHID* which represents the corresponding channel (the owner of a channel also has to subscribe in order to receive this information and to be able to distribute content via the channel).

5.3.3 Sharing Information via a Channel

Every server construct in which the agent `SHARINGMGR` is embedded is always located in an ambient which represents a particular channel and whose name corresponds to the identifier of the channel. In order to be able to perform its task, it is required that each instance of `SHARINGMGR` knows by some static nullary function called *myChId* the name of the ambient in which it is executed.

⁸This is the only way how an ASM agent can make changes in the ambient tree hierarchy contained by dynamic-derived function *curAmbProc* [31].

SHARINGMGR is a parameterized ASM agent, which expects the following arguments: *sndr* is the registered name of the sender, *rcvr* is either the registered name of a receiver or an asterisk “*,” and *info* is either the content of *ShareInfoMsg* or the description of a shared service operation in *ShareSvcMsg*. The last three arguments are not used in the case of the message *ShareInfoMsg*, and the value **undef** is assigned to each of them by the surrounding server construct. In the message *ShareSvcMsg*, *o* denotes the unique identifier of the service operation that *sndr* is going to share, *argsP* denotes the arguments of *o* that *rcvr* can freely modify if he or she calls the operation, and *argsF* denotes that part of the argument list of *o*, whose value is fixed by *sndr*.

The agent first generates a new and unique operation identifier for the service operation *o* (if *o* is not equal to **undef**). This new identifier which is stored in the nullary location function *shOp* will be announced to the channel member(s) specified in *rcvr*. In the next step, the agent checks whether the *sndr* is a registered member of the channel. Then if the given value of *rcvr* is equal to “*,” the agent broadcasts the corresponding message(s) to all members of the channel. Otherwise, if the value of *rcvr* corresponds to the name of a particular member of the channel, the agent sends the corresponding message(s) only to him or her.

In the case of the processing of *ShareInfoMsg*, the agent sends to the member(s) specified in *rcvr* the message *sharedM_{content₁}* (for its definition, see Table 3), which contains the sender *sndr* and the shared information *info*.

In the case of the processing of *ShareSvcMsg*, two ambient constructs are created by NEWAMBIENTCONSTRUCT. The first one is the message *sharedM_{content₂}* (for its definition, see Table 3) and it is sent to the member(s) specified in *rcvr*. It contains the sender *sndr*, the new operation identifier *shOp*, the list of public arguments *argsP*, and the informal description of the shared operation denoted by *info*.

The second ambient construct is the plot PLOT_{shOp} (for its definition, see Table 3) enclosed by the ambient *newPlot* and equipped with some additional ambient actions (see the underlined capabilities in the definition of *sharedPlot* in Table 3) which move the entire construct into the user area of the channel member(s) specified in *rcvr*, where the plot will be accepted by the term **!ALLOW newPlot**.

Table 3 Definitions of ambient constructs used by the ASM agent SHARINGMGR

<i>sharedM_{content_i}</i>	$\equiv \text{returnValue}[\text{OUT } n.\text{OUT myChId}.\text{IN } \text{UID}.\langle \text{cname}, \text{client}, \text{content}_i \rangle]$
<i>content₁</i>	$\equiv \{ \text{“sender:” } sndr, \text{“content:” } info \}$
<i>content₂</i>	$\equiv \{ \text{“sender:” } sndr, \text{“operation:” } shOp, \text{“arguments:” } argsP, \text{“description:” } info \}$
<i>sharedPlot</i>	$\equiv \text{newPlot}[\text{OUT } n.\text{OUT myChId}.\text{IN } \text{UID} \mid \text{PLOT}_{shOp}]$
PLOT_{shOp}	$\equiv \text{SERVER}_{op}^s shOp.trigger_o$
<i>trigger_o</i>	$\equiv (v \text{ tmp})$ $(client, argsP).(\text{OUT } \text{UID}.\text{IN } \text{UID}_{sdr}.s \text{ BE request } $ $o[\text{ ALLOW } op.\langle tmp, (argsP \setminus argsF) + argsF \rangle] $ $tmp[\text{ ALLOW } output \mid \text{CID}[(o, c, a).\text{OUT } \text{UID}_{sdr}.$ $\text{IN } \text{UID}.tmp \text{ BE returnValue}.\langle shOp, client, a \rangle]])$

The execution of the shared service operation $shOp$ can be requested in a usual $RequestMsg$ as normal service operations. The $PLOT_{shOp}$ is a plot, which can accept service operation requests for $shOp$ several times. It is a special plot, because instead of triggering the execution of $shOp$ as in the case of a normal operation, a normal plot does (see [30]); it converts the original request to another request for operation o by applying the term $trigger_o$. This means that it substitutes the operation identifier o for $shOp$, it completes its argument list with $argsF$, and it forwards the request for o to the user area of the user $sndr$ who actually has right to trigger the execution of the operation o .

To the new request, the name-restricted ambient tmp is attached (see its definition within the definition of $trigger_o$ in Table 3), whose purpose is similar to the communication endpoints of registered clients. Namely, it is placed into the user area of $sndr$ temporarily and it is responsible to forward the outcome of this particular request from the user area of $sndr$ to the user area of the user who initiated the request. It is beyond the scope of this chapter to present a reduction on how a particular request for a shared operation is processed in our model, but we refer to [29] for more details.

6 The Specification of the Identity Management Machine

The identity management machine (IdMM), first presented in the paper [115], is a privacy-enhanced client-centric identity meta-system based on the concept of abstract state machines. The IdMM provides a “proxy” between a client and cloud-based identity management solutions “translating” the protocols used by the client to manage identities to a set of protocols used by cloud providers. The IdMM can then authenticate a user to a given cloud service as well as manage any private identity-related data stored on the cloud.

As mentioned in Sect. 3.2, the IdMM is composed of six agents. Figure 5 details the overall architecture of the system. The six agents, first introduced in the paper [116], define the various interactions needed for the authentication, de-authentication, and attribute synchronization for cloud services, based on the abstract functions described in Fig. 6. In this section, we will give an overview for of the agents: the core, client, cloud, and user agents. We will then detail a proof-of-concept implementation previously presented in [119]. Since the specification of the agents is still an ongoing task, we will end this section with a description of future work on the IdMM.

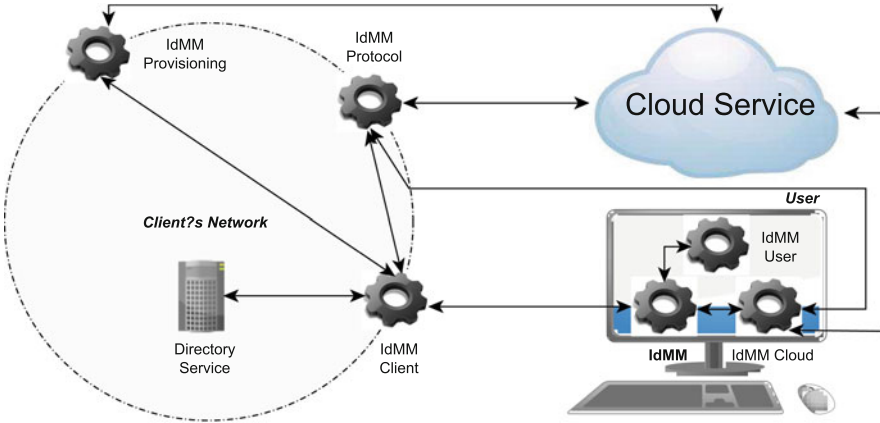


Fig. 5 IdMM architecture

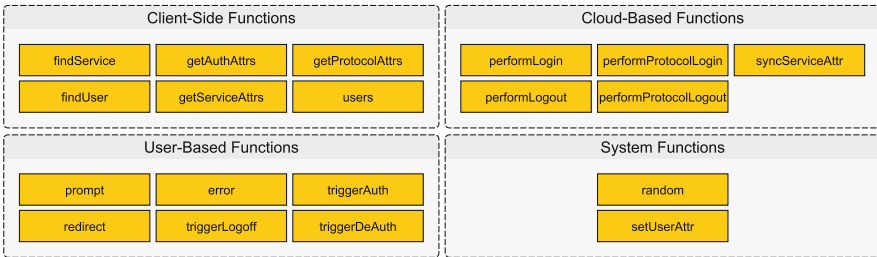


Fig. 6 Abstract functions

6.1 The IdMM Core Agent

The core agent for our client-centric solution is described in the paper [115]. The agent has nine states (Fig. 7). The initial state of the IdMM is *UserLogin*. While in this state, the user is prompted to input his or her credentials and is subsequently authenticated to the machine. If a user cannot be authenticated, the machine halts with the appropriate error. When a user wants to log out, the event triggered will set the state to *UserLogout*. This will log the user out of the machine, set the state to *UserLogoutService* which logs the user out of every single service he or she was connected to, and halt the execution of the machine. The termination of the machine occurs in the *halt* state. If an error exists, the appropriate message will be displayed to the user.

When the user wants to use a specific service, he or she will specify the service’s uniform resource identifier (URI). The event triggered by this action will set the state to *ServiceLogin*. In the *ServiceLogin* state, the machine attempts to find the matching service given the URI. If no services can be found, an error message is triggered. If a service is found, the state will be set to *AuthorizeLogin*. In case the

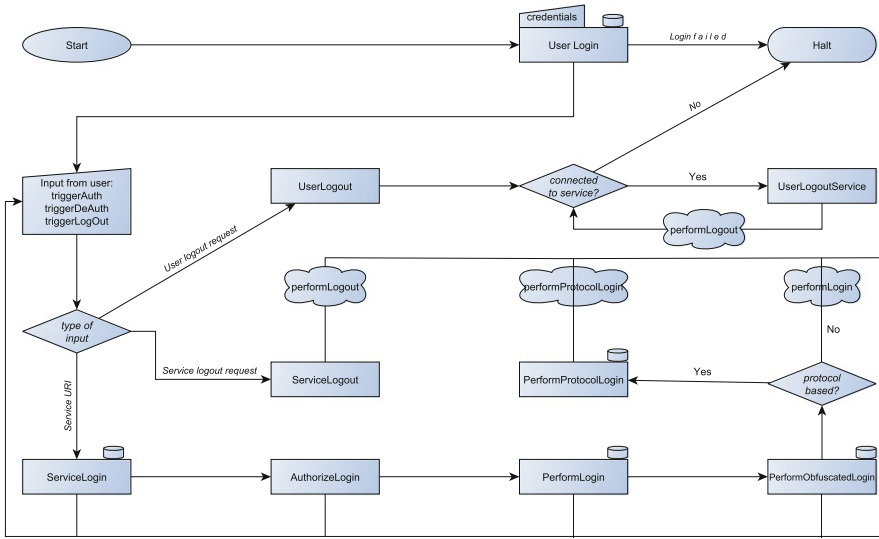


Fig. 7 IdMM flow

user is already connected to the service, he or she will be redirected to the given URI. The authorization is done by the *AuthorizeLogin* state. If the user has the appropriate access rights, the state is set to *PerformLogin* where the machine checks the type of identity required and sets the state to *PerformObfuscatedLogin*. If the service supports protocol-based authentication, then *PerformObfuscatedLogin* will switch to *PerformProtocolLogin*, which will perform the authentication based on a given protocol. In case the service does not support protocol-based authentication, the machine will perform the authentication with the given identity. The machine switches to the *ServiceLogout* state when a log-out event is triggered. In this state, the IdMM searches for the matching service and performs the log-out.

To describe the rules for each state, we make use of abstract functions (Fig. 6). The client-side functions, cloud-based functions, and user-based functions will be further refined by describing the $IdMM_{Client}$, $IdMM_{Cloud}$, $IdMM_{Protocol}$, and $IdMM_{User}$ agents. To ease this process, we keep the underlying communication layers abstract.⁹ The system functions include two important functions: *random* and *setUserAttr*. In order to handle an obfuscated identity, we use the function *users* to select a list of the obfuscated identities contained in the client’s directory. We would then choose a random identity from this list and use it to authenticate the required cloud service. At present, we propose that the function *random* be restricted to only this requirement. More research can be conducted in this topic to further enhance

⁹In the paper [119], for example, we have included the $IdMM_{Client}$ agent as part of a Tomcat server. The communication with the core agent is done via the Google Web Toolkit RPC framework.

privacy. For instance, we might choose not to include identities that have been previously used with the given service. If the service requires a partially obfuscated identity, we use the function *setUserAttr* to replace some of the obfuscated values in the identity with real values that correspond to the current user.

6.2 The IdMM Client Agent

The $\text{IdMM}_{\text{Client}}$ agent is responsible with the interaction with the client's directory service. It retrieves any relevant information from the directory service and converts it to the IdMM's data structures. To achieve this, we have further refined the client-side functions presented in Fig. 6. Apart for the *getAuthAttrs* function, the refinement involves introducing a new submachine for each client-side function.¹⁰

The refinement of the client-side functions is described in the paper [117]. To allow a seamless adoption of the IdMM, we make an abstraction with respect to the client's directory. The refinement of the client-side functions yielded an interface (*client interaction interface*) of 20 directory-dependent functions (Fig. 8). In addition to the interface, the refinement of the client-side functions also yielded a list of parameters for each of the submachines. These parameters include information about the client's directory (location, authentication mechanism, credentials) as well as information about the structure of the client's directory. In the paper [117], we illustrated how to implement a client interaction interface for an ApacheDS server.¹¹

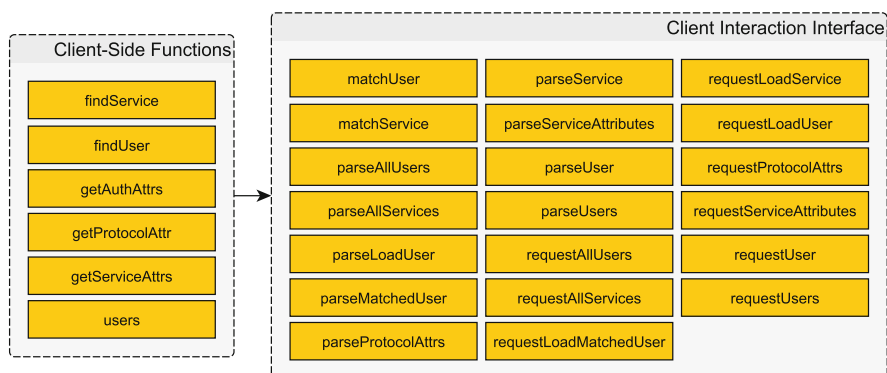


Fig. 8 Client-side function refinement

¹⁰For the purpose of this paper, we will not detail each submachine. The refinement and description of the client side functions are described in the paper [117].

¹¹ApacheDS [11] is a Java-based embeddable directory server certified by the Open Group as LDAPv3 (Lightweight Directory Access Protocol) [101] compatible.

We used the Novell LDAP Classes for Java API [84] to facilitate the interaction with the ApacheDS server.

6.3 The IdMM Cloud Agent

The $IdMM_{Cloud}$ agent is responsible for the interaction with any cloud service. Its purpose is to authenticate or de-authenticate a user to/from a given cloud service and, when needed, to perform attribute synchronization (see Sect. 3.2.1). To achieve this goal, the cloud-based abstract functions presented in Fig. 6 must be further refined. At present, we have the specifications for only the direct and obfuscated client-to-cloud interaction scenarios presented in Sect. 3.2. Our current work entails the understating and specification of the protocol interaction as detailed in Sect. 10. As such, the refinement of the functions *performLogin*, *performLogout*, and *syncServiceAttr* is achieved by introducing the $IdMM_{Cloud}$ submachine as presented in the paper [120].

Since the direct interaction scenario takes into consideration the fact that different cloud providers will have different authentication systems, we have introduced the concept of a cloud plug-in (Fig. 9). This interface represents a set of functions that will be used for authentication, de-authentication, and attribute synchronization on various cloud services. For each cloud service used, an implementation of this interface must be provided. We consider two distinct ways for authentication, de-authentication, and attribute synchronization. In case the service provider already offers an API for this purpose, we simply use the functions *makeAPIAuthentication*, *makeAPIDeAuthentication*, and *makeAPISync*. If the service provider does not have such an implementation, we use a custom-created one marked by the *generic* functions. For example, we use the functions *requestAuthParameters* and *parseAuthParameters* to obtain all parameters used for the authentication process. We then use the functions *requestGenericAuthentication* and *parseGenericAuthentication* to make the actual authentication process.

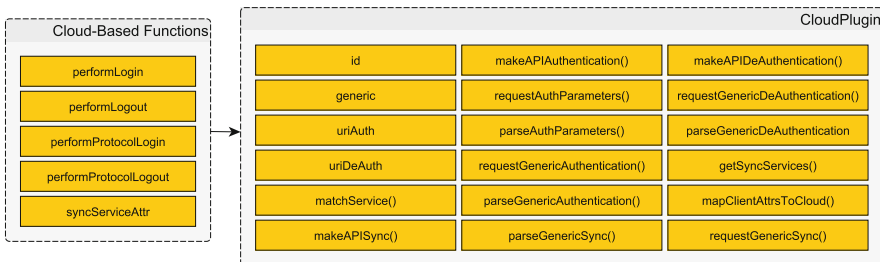


Fig. 9 Cloud plug-in

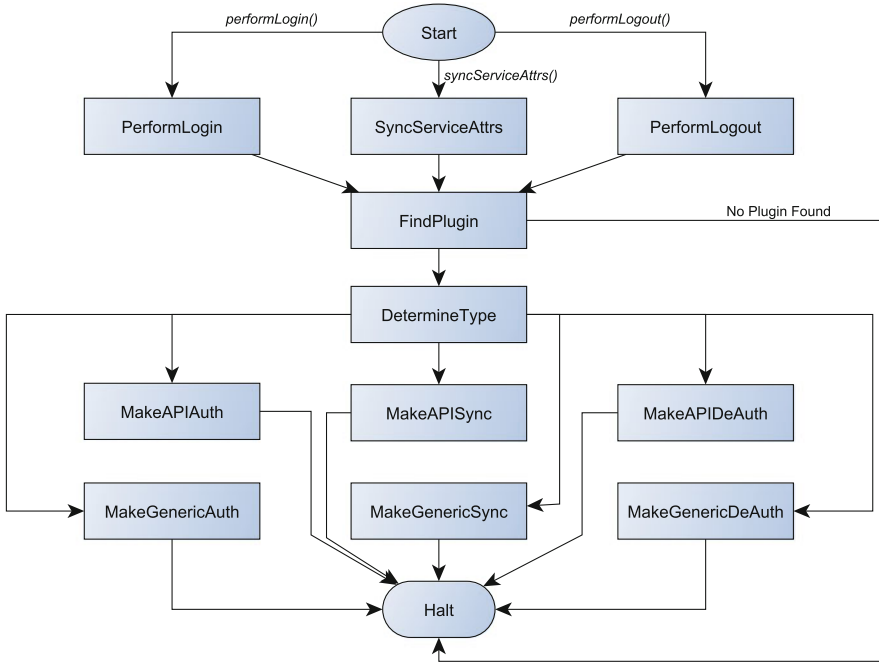


Fig. 10 IdMM_{Cloud} flow

The rules for the IdMM_{Cloud} submachine are presented in the paper [120]. The initial state of the machine depends on the function being called. For the function *performLogin*, the initial state will be *PerformLogin* (Fig. 10). Similarly, the initial states will be *PerformLogout* and *SyncServiceAttrs* for the functions *performLogout* and *syncServiceAttrs*. While in these states, the appropriate values of the machine's dynamic frame are set and the state is set to *FindPlugin*. In this state, the machine attempts to find a plug-in that matches the provided service. If no plug-in is found, the machine halts. When a plug-in is found, the state of the machine is set to *DetermineType*. The rules for this state determine which of the *MakeAPIAuth*, *MakeAPIDeAuth*, *MakeAPISync*, *MakeGenericSync*, *MakeGenericAuth*, or *MakeGenericDeAuth* states will be chosen based on the type of the request and whether the service offers an API. While in the aforementioned states, the actual authentication, de-authentication, or attribute synchronization processes take place via the use of the *CloudPlugin* functions presented in Fig. 9.

6.4 The IdMM User Agent

The IdMM_{User} agent is responsible for the interaction with the user. It handles inputs from the user and displays the appropriate messages. This goal is achieved by implementing the user-based functions presented in Fig. 6. Since the implementation of these functions is software dependent, we opted to keep the functions abstract.

The function *prompt* is used to prompt the user into typing his or her credentials and returns a list of attributes representing the credentials. The function *error* is used to notify the user of an error that occurred during the authentication process. The function *redirect* is used to redirect the user to a specific URI, specified as a parameter. The event triggered by the user entering a URI will call the function *triggerAuth* which changes the state of the IdMM core agent to *ServiceLogin*. When the user wants to de-authenticate from a service, the underlying event will call the function *triggerDeAuth*, causing the state of the core agent to change to *ServiceLogout*. The de-authentication from the IdMM is done via the *triggerLogoff* function, which sets the state to *UserLogout*. During the implementation phase described in the paper [119], we became aware that some Web-based applications do require the user to enter dynamically generated data for the sole purpose of disallowing automated requests. Most often, this is achieved via captcha messages. As such, we were forced to introduce a new user-based function, *String captcha(Object message)*, to allow the user to input captcha messages.

6.5 A Proof-of-Concept Implementation

In the paper [119], we described a proof-of-concept implementation for the IdMM focusing primarily on software-as-a-service (SaaS) platforms. The implementation follows the architecture described in Fig. 5. We used an ApacheDS [11] directory service running in a cloud-based environment on a private virtual machine. The implementation for the IdMM_{Client} agent is running as part of an Apache Tomcat [12] server. The communication between the IdMM_{Client} agent and the ApacheDS service is done via the use of the Novell LDAP Classes for Java [84]. The client interaction parameters described in Sect. 6.2 are stored in a *.properties* file (as opposed to an XML file as described in the paper [117]). The Tomcat server hosting the IdMM_{Client} agent is running on a public virtual machine in the same cloud-based environment as the ApacheDS server.

The IdMM_{Core} , IdMM_{Cloud} , and IdMM_{User} agents are running as part of a Google Chrome browser extension. Since the specification of Google Chrome extensions [58] entails the use of JavaScript, as opposed to the usage of Java for the IdMM_{Client} , we opted to use the Google Web Toolkit framework [59]. By using this framework, we can restrict the programming language of the implementation to only Java, leaving the compilation of the code to JavaScript down to the GWT framework. The communication between the IdMM_{Core} agent and the IdMM_{Client} agent (for

the execution of the client-side function presented in Fig. 6) is achieved by using GWT's own RPC framework. The credentials returned by the user-based function *prompt()* as well as the address of the $\text{IdMM}_{\text{Client}}$ are stored in encrypted format in the browsed local storage environment. The user may modify the values via the use of the extension's option page. All communication between the client's directory and the $\text{IdMM}_{\text{Client}}$ agent, $\text{IdMM}_{\text{Core}}$, and $\text{IdMM}_{\text{Cloud}}$ agents is encrypted. Any privacy-related data stored in the dynamic frame is encrypted as well.

A demo of the implementation can be found in [118].

7 Cloud Content Adaptivity Specification

The adaptivity solution, first presented in [43], uses a WA wrapped inside a middleware system to adjust cloud services to client's device. This research includes the creation of ASM ground models in a way to reflect the requirements and to serve as a basis for implementation. The next subsection introduces the system architecture of the proposed solution.

7.1 The System Architecture

The system architecture illustrated in Fig. 11 shows how the cloud, the middleware, and the client (here represented by his or her devices) are interacting. The client's point of connection is a WA which should adapt itself, on the fly, to different devices (smartphones, tablets, laptops, and desktop computers) and different browsers. The interaction between the client and the cloud is done through a middleware software. The WA represents the front-end and the middleware the back end. After a successful log-in, the application will display a list of cloud services corresponding to the user's credentials. By selecting a service, a request is sent to the middleware, which will forward it to the cloud and wait for the corresponding answer. Meanwhile, a device profile will be created, using the third-party tools and frameworks.

For detecting the device properties, the *Modernizr* framework is used by creating JavaScript tests. These tests will be executed on the client side and the corresponding cookie variable will be updated, sending the information to the server side, where the session and the local database will also be updated. When the features cannot be detected using *Modernizr*, then a device detection database tool can be used (e.g., Wireless Universal Resource FiLe (WURFL)). If the user uses the cloud services from the same device several times, then the device information can be read from the local database, without accessing the device detection database or writing again the JavaScript tests. Even when the information is locally available, the session is still used, because of optimization purposes. Another reason for saving the information locally is also the client-client interaction. One client could choose to send the output

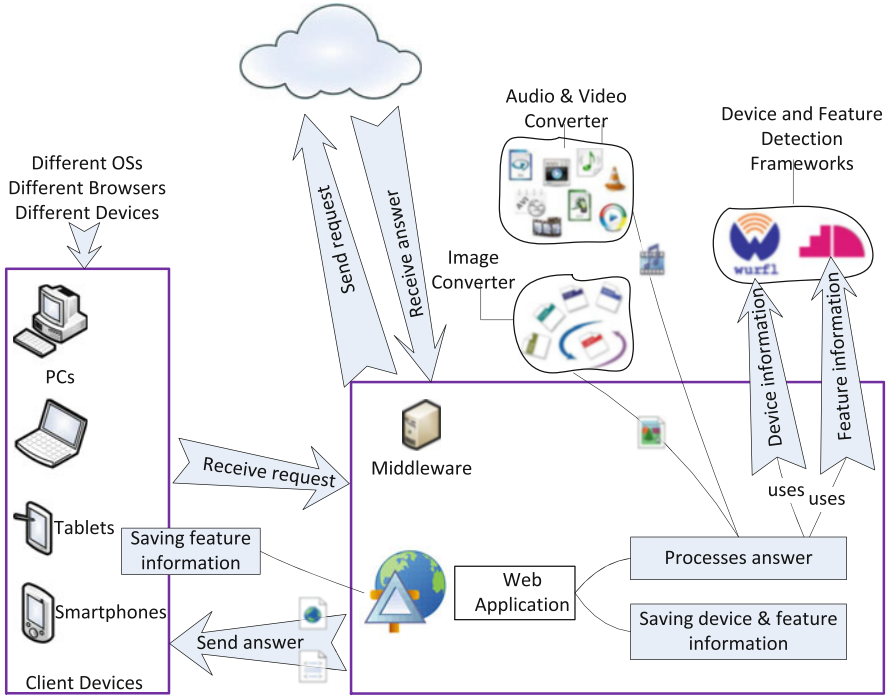


Fig. 11 System architecture

of a service from cloud to another client, which is logged in from another device, which means that sending the answer from the cloud not only to the device for which we have the profile in the session but also to another device for which we can read the information from the database.

The answer that arrives from the cloud is parsed and processed using the device information. When images or videos are involved, their format is checked, and if it is not accepted by the device, then third-party tools are used to transform the media to the corresponding format. When the adaptation is finished, the message is sent back to the device. In case some JavaScript tests exist, they will be executed on the client side for getting the new information; afterward, the cookie is updated. When the loading finishes, the message is displayed on the device.

The middleware acts like the virtual provider described in [9]. In our case, we leave out for now the sending and receiving part and concentrate only on the adaptation.

7.2 ASM Ground Models

In order to reflect the WA intended behavior, the ASM method was used for building the ground models. The main components of the system were modeled such that the refinement of their abstract models would be a description of the future implementation.

Further on, the models including only aspects with respect to content adaptation and displaying are presented. Most of the work is done on the server side using the information discovered on the client side.

7.2.1 Display Output Agent

Figure 12 illustrates the ASM ground model that describes the client's device activity. There are three states through which the agent goes, *Waiting for message* (the initial state), *Execute client tests*, and *Displaying the message* (the final state).

The agent waits in the initial state until a message comes from the server. The messages sent by the middleware to the client are saved in a queue. When a message becomes available in the queue, the agent executes the macro *Decrypt message*. After the decryption of the message, the flow goes further with the condition *Client tests available*. If JavaScript tests (which verify the device properties) are available, then the agent goes to the state *Execute client tests*. This is a durative action (there is an interval of time between starting and ending the execution) which is executed internally by the browser. When the JavaScript execution finishes, the agent retrieves the new device information and updates the cookie, in this way communicating to the server the new values of the device properties. The flow takes the same route, as when no client tests would exist. The guard *Extra resources* verifies if extra

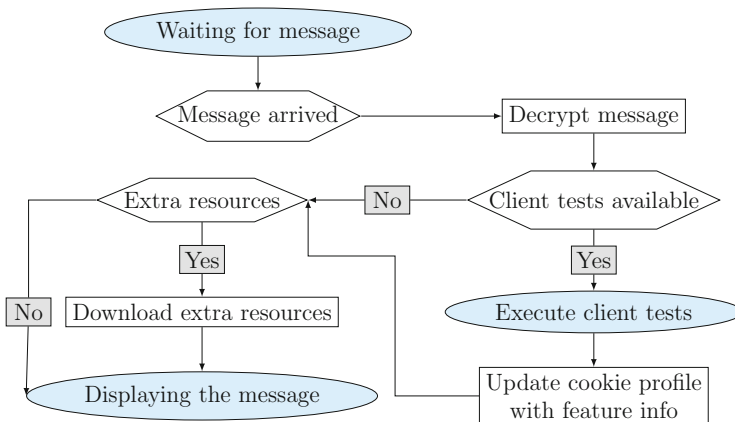


Fig. 12 Display output ASM

images and/or videos are necessary. In a positive case, the system downloads them (the abstract macro *Download extra resources* does this action). The agent’s state changes, reaching the final one, *Displaying the message*.

7.2.2 Receive Request Agent

Figure 13 displays the ASM ground model for receiving the client’s request. The algorithmic idea consists in having only the initial and the final states: *Waiting for requests* and *Waiting for answers from Cloud*. Again, a queue is used to store the requests sent by the client. When a client’s request arrives, it is then forwarded to the cloud (this is fulfilled by the *Send requests to Cloud* macro), and in parallel, the agent verifies if the session or the cookie contains information regarding the device which sent the request.

If the condition *Device info available in the session/cookie* is not met, then we are dealing with the first request sent during that session. Using the guard *Device profile available on the server*, the agent verifies if this is the first time that the user logs in from the corresponding device. If this is the case, then the agent retrieves the device details from the local database by executing the *Retrieve the local device profile* macro. After retrieving the device details, another macro is executed to *Update session/cookie with device info*. The information is saved on the session and on the cookie to ease the communication between client and server agents and to optimize the process regarding the verification of device properties. Whenever a new JavaScript test is executed, the cookie and the session variables are updated using the test’s result. The state is changed and the agent waits for answers from the cloud.

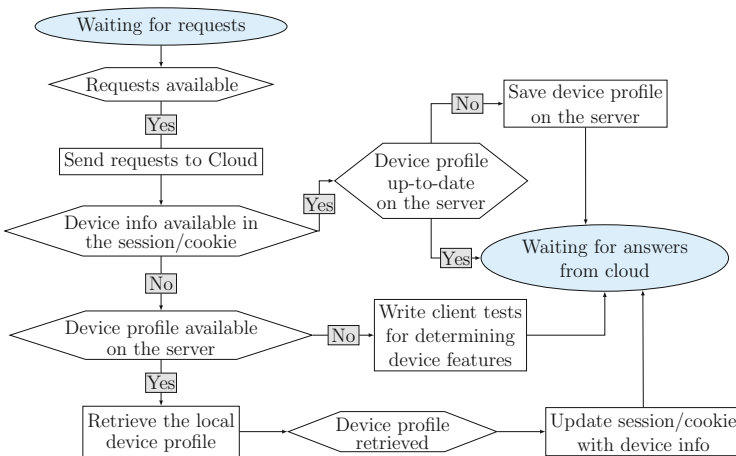


Fig. 13 Receive request ASM

If the guard *Device profile available on the server* is not satisfied, then the client's tests for retrieving the values of all device properties that are necessary for code adaptation are written. The macro *Write client tests for determining device features* is responsible for the previous action. In the same time, the state is changed to *Waiting for answers from Cloud*.

By writing JavaScript code together with *Modernizr*, the client's tests are created in order to determine the device and browser capabilities. The tests will be written on server side and executed in background on the client side. After the execution finishes, the answer is sent back to the server. Information regarding JavaScript interpretation using ASMs is provided by Börger et al. [27]. Using the information provided in that article, the macro *Update session/cookie profile with feature info*, which can be seen by the reader in Fig. 12, deals with the description of the client's tests. Starting from the basic client's tests, new tests could be inserted if, while parsing the answer sent by the cloud, new device properties are involved.

Another way to follow is when the information is already available in the session, which means that the request in case is not the first one coming from the client. Using the guard *Device profile up-to-date on the server*, the agent verifies if the local database already contains the device information. If no modification is necessary, the final state is reached. In the contrary case, when device properties have to be created or updated, the macro *Save device profile on the server* is executed. At this point, the agent also reaches the final state.

7.2.3 Receive and Process Answer Agent

The algorithm presented in the ground model from Fig. 14 describes what the agent does with the answer sent by the cloud in order to adapt it to the device. The agent finds itself in four different control states: *Waiting for answers from Cloud*, *Filter and adapt content*, *Message format transformed*, and *Send answer to client*. The starting control state is *Waiting for answers from Cloud* and the final state is *Send answer to client*.

The agent waits for the messages having the state set to *Waiting for answers from Cloud*. The messages sent by the cloud are saved in a queue which is checked by the guard *Message available*. Whenever a message is available, a verification is done to see if its format is supported by the browser installed on the device (this check is contained by the guard *Message format supported*). If the message is not written in Hypertext Markup Language (HTML) or Extensible Hypertext Markup Language (XHTML), then the macro *Transform the message format (html/xhtml)* is executed and the agent's state is changed to *Message format transformed*. If the message's format can be displayed in the browser, we go further on with checking if the device details are available, using the guard *Device information available*. What is this guard actually doing? It checks to see if the device profile is available in the session. If the device information is not already available, then the flow goes on with the next condition without adapting the HTML code. In this case, the page will be

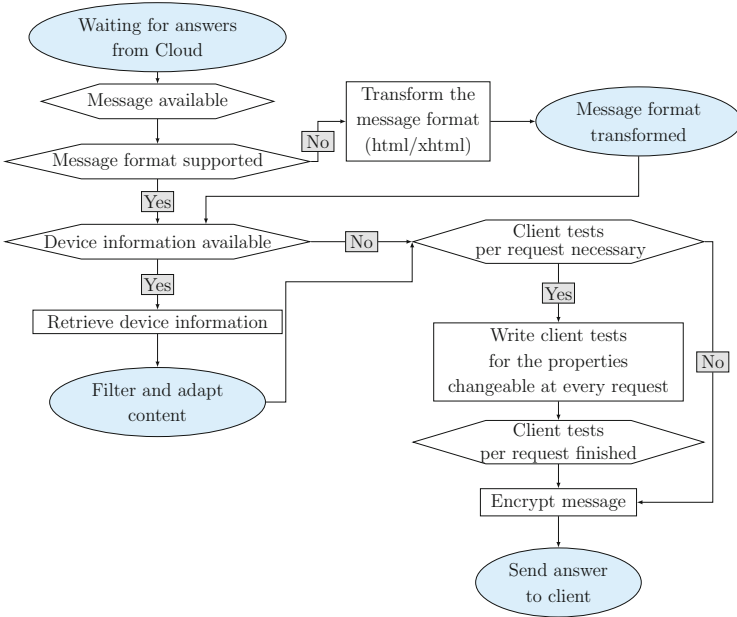


Fig. 14 Receive and process answer ASM

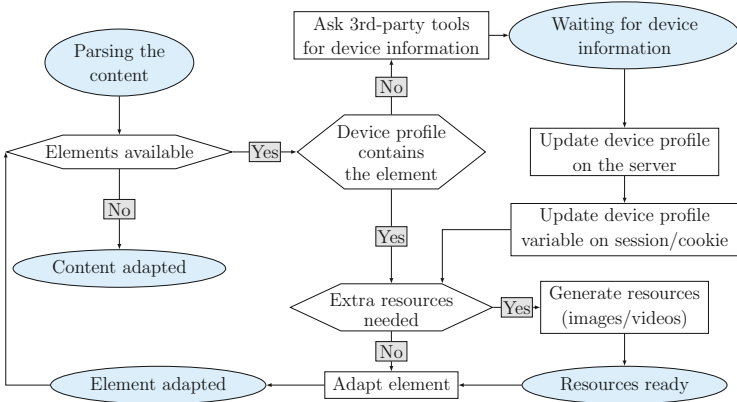


Fig. 15 Filter and adapt content ASM

displayed on the device in its original state, containing only the adjustments done automatically by the browser.

If the device information is available in the session, then using the macro *Retrieve device information*, the corresponding information is retrieved and the agent’s state is set to *Filter and adapt content*. This action is extended in the model displayed by Fig. 15. When the adaptation finishes, the agent reaches the same condition as the one reached when the device information is not available.

The guard *Client tests per request necessary* checks if an extra JavaScript code has to be inserted in order to test the device capabilities which can change by every request. An example is the Global Positioning System (GPS) location property. GPS information is something that could change each time the user makes a new request, because he or she might change his or her position (we can imagine that the user is moving, being in a train or in a car, when accessing the cloud). If there are no tests per request required, the agent has one more step to do before going to the final state, which is *Encrypt message*. When this macro is executed, the control state *Send answer to client* is reached. The macro *Write client tests for the properties changeable at every request* is executed if the corresponding tests are necessary, and the guard *Client tests per requests finished* verifies if this execution finished and if the agent is ready for encryption. The same flow as for the “No” branch follows; the macro *Encrypt message* is executed and the final state is reached.

7.2.4 Filter and Adapt Content Agent

For adapting the content coming from the cloud, a general HTML/XHTML parser has to be designed. As a basis for this, the paper [56] can be used. In [56], a multi-agent ASM formal model is described, which shows the browser behavior and how its components interact.

Figure 15 displays the algorithm of the *Filter and adapt content* model, which was mentioned as a control state in the previous ground model (Fig. 14). In this ASM ground model, the agent can reach five control states: *Parsing the content*, which is also the initial state, *Waiting for device information*, *Resources ready*, *Element adapted*, and the final state *Content adapted*.

The adaptation algorithm starts with the parsing of the HTML content of the answer which came from the cloud. Each HTML element is identified, while parsing the content and the guard *Elements available* verifies if the algorithm still has to check and adapt other HTML elements. If this is the case, then the agent goes to the next condition *Device profile contains the element*. If the device profile saved in the session does not contain information with respect to the chosen element, then a device detection database third-party tool is used. A query containing the specific element is sent to the device detection database. When the agent reaches the macro *Ask 3rd-party tools for device information*, the state changes to *Waiting for device information*. The agent waits until this durative action is executed by the device detection database. When the information is returned to the agent, two macros are executed in parallel: *Update device profile on the server* and *Update session/cookie with device information*. By updating the information in the session and in the local database, we will not have to query again the third-party tool regarding the same HTML element for the possible future requests. Nearby these two actions, the guard *Extra resources needed* is also fired.

One of the existing device detection databases can be used, without creating an own database, because this would cost a lot of time, mainly because of the database maintenance.

The agent executes the *Extra resources needed* guard for verifying if the element's (e.g., an image or a video) format is supported by the browser running on the current device. In case the format is not applicable, the macro *Generate resources (images/videos)* is executed and the control state is updated to *Resources ready*. The next step is the same as the one when no extra resources are needed.

The *Adapt element* macro does the job of changing the HTML element corresponding to the device profile. If, for example, a very complex list should be displayed on a smaller device, on which the user should scroll too much for seeing the information, then the list should be changed by eliminating the not-so-important information and make it expandable, such that the user could easier read the information and reach faster the information which is interesting for him or her. By detecting the device properties, the developer discovers which types of elements are suitable for the device and can also change the format correspondingly. Reaching the execution of this last macro, the state corresponding to each element is set to *Element adapted*. Changing the state of each element is like a flag which is set on each element. The agent goes to the final state, *Content adapted*, when all elements reached the state *Element adapted*.

7.3 The Use of ASM Ground Models

In the previous section, the WA's constraints/assumptions were expressed by the creation of ASM ground models. The fact that the requirements change based on the device that is querying the cloud services requires the development of a flexible and extendable piece of software by using successive refinement steps that adapt the abstract models to the changing requirements.

For reaching the compilable code, the next phase after the creation of the ground models is the definition of the macros by using pseudo-code-like descriptions. Typically, there are necessary more steps to reach the code, because the design phase starts with a not-so-precise model and gradually more details are introduced with each step, taking into account the requirements of the system [23]. Below is an exemplification of how the *Display Output Agent* ground model from Fig. 12 can be refined by describing the agent's signature and the ASM macros.

Below follows the agent's signature. The $ctl_state = \{ \textit{Waiting for message}, \textit{Execute client tests}, \textit{Displaying the message} \}$ variable represents the agent's control states and has as initial value the state *Waiting for message*. The queue $messages(self)$ contains the messages sent by the middleware; its initial value is the *empty set*. The $cookie(deviceProfile) \in COOKIE$ variable stores the device profile, where the set $COOKIE$ contains $key \times value$ elements (initial value = *undef*). $headPage: messages(self) \rightarrow html$ is a function that returns the first HTML text sent by the middleware. Its initial value is *undef*. *Modernizr* is a set that contains the features (and their corresponding values) tested by using the Modernizr framework. Using the agent's signature, the macros can be defined as follows.


```

CLIENTDISPLAYOUTPUTMACROS =
  if MESSAGEARRIVED then
    DECRYPTMESSAGE
    if CLIENTTESTSAVAILABLE(headPage(
      messages(self))) then
      ctl_state := executeClientTests
      UPDATECOOKIE
    end if
    if EXTRARESOURCES then
      DOWNLOADEXTRARESOURCES
    end if
    ctl_state := displayingTheMessage
  end if

```

Where

```

MESSAGEARRIVED = (messages(self) ≠ empty)
DECRYPTMESSAGE - abstract
CLIENTTESTSAVAILABLE(page) =
  (∃t ∈ htmlTags tagName(t) = "SCRIPT" and
   CONTAINSMODERNIZRTESTS(t))
CONTAINSMODERNIZRTESTS(scriptTag) =
  if hasAttribute(scriptTag, src) then
    let url = valueOfAttribute(scriptTag, src) in
      ∃c ∈ contentOfFile(url) c = "Modernizr.addTest"
  else
    ∃c ∈ contentOfTag(scriptTag)
      c = "Modernizr.addTest"
  end if
EXTRARESOURCES = (extraResources ≠ empty)
DOWNLOADEXTRARESOURCES - abstract
UPDATECOOKIE =
  if ctl_state = executeClientTests then
    for all f ∈ self.Modernizr do
      insert f into self.cookie(deviceProfile)
    end for
  end if

```

CLIENTTESTSAVAILABLE macro verifies if there are Modernizr tests defined in the JavaScript part of the page. There are two possibilities to declare the client's tests, directly in the SCRIPT section of the page and through another JavaScript file. The "Modernizr.addTest" method (made available by the Modernizr framework) can be used either in the content part of the SCRIPT tag or inside the JavaScript file.

UPDATECOOKIE macro parses all the features (*key* × *value* pairs) tested using the Modernizr framework and creates the agent's cookie.

Following the example given above, each of the ground models defined in Sect. 7.2 could be refined to pseudo-code and afterward translated to CoreASM.

8 The Problem of Verification

Although our work is focused only on the specification of some novel solutions concerning client-cloud interaction at present, the validation and the verification of our (ground) models have been planned as our future work. Hence, we devote the following section to this subject.

In the context of software systems, formal verification can be regarded as a decision problem [105]. Namely, if it is given a programming language L and a formal language F suitable to express properties of programs in L (e.g., completeness, soundness, compactness, liveness, and safety and halting problems), the problem of verifying L programs against F properties can be considered as follows [105]:

- Given a program $\Pi \in L$ and a property $\varphi \in F$, decide whether for every input \mathcal{I} appropriate for Π , Π on input \mathcal{I} satisfies φ .

The decidability of this problem depends on the expressiveness of both L and F . Although there exists some artificial subset of formal languages which are extended with some (first-order) logics and for which the above problem is always decidable (e.g., sequential nullary ASMs conjugated with a first-order branching temporal logic [105]), their computational power and expressiveness are limited.

Verification in process calculi (e.g., ambient calculus) and in state transition systems is often done by checking bisimulation equivalence [104], where it is investigated whether the associating systems behave in the same way in the sense that one system simulates the other and vice versa. A more complex process representing an implementation is shown to be bisimilar to a simpler process representing a specification. The simpler process should be so clear that it can be regarded as satisfying correctness requirements in an intuitive sense, not with rigorous mathematical proof.

This procedure may be not reliable or not applicable at all for many cases in general (intuitive checking of correctness of *formal ground model* is often nontrivial and a very error-prone method). Hence, for verifying correctness properties for algorithms specified by these formal languages, some (temporal) logics are required, such that some model checking verification can be applied. There are model checking algorithms for process calculi with calculus style specification logic [17, 49], but such a logic often is very complicated to describe or understand, and sometimes results are not obtained in reasonable amount of time even for the proofs of very simple correctness requirements.

In some other cases, the aim of the coupling of a formal language and such a special logic is to address only the verification of specific aspects or features in the specified systems. For instance, *Cloud Calculus* [67] applies ambient calculus

to specify topology of cloud systems and firewall security rules such that it is able to verify whether the global security policy after the virtual machine migration is consistently preserved with respect to the initial one.

In the case of the formal model of our integrated system discussed in this chapter, which is given in terms of our two-abstraction-layer approach mentioned above, verification can be reduced to pure ASM verification, since the definitions which are given in terms of ambient calculus can be replaced with ambient ASM rules which in turn can be translated to normal ASMs.

Although ASMs can be seen merely as a specification formalism, strictly speaking, ASMs constitute a computation model on structures. The program of an ASM is like the program of a Turing machine, a description of how to modify the current configuration of a machine in order to obtain a possible successor configuration. Since ASMs are computationally complete—they can calculate all computable functions—the problem given above is, in its full generality, undecidable.

The advantage of ASMs is that they are close to logic, which makes the overall design easily amenable to well-understood mathematical techniques. Essentially, the mathematical foundation of ASMs supports the formal verification of dynamic systems designed by means of ASMs.

For the verification of the properties of ASMs, mechanical theorem proving (e.g., Isabelle [24, 89], KIV [93], PVS [51, 54], etc.) as well as model checking systems [106, 121] can be applied. For these, some logics were also developed [107], which are tailored for ASMs in terms of an atomic predicate for function updates.

After the correctness of an ASM (ground) model is proved, we can apply stepwise refinement method [22] to extend the model and to prove in each step that the new model preserves the same properties, which the predecessor model possessed. The design of complex systems is typically organized as a series of refinement steps, where the final goal is to minimize the gap in reasoning between the refined formal model and a particular implementation.

In Sects. 8.1 and 8.2 below, it is presented shortly which correctness properties for Web application are assumed and how could ASM ground models help, respectively, and which solutions are available for the client- and server-side adaptation.

8.1 Correctness of Web Application

WAs do not have a precise definition or a precise model to follow. This happens because they are related to different standards (which can be incomplete or bad documented) and implementation frameworks [27]. They are using a huge number of diverse technologies and there is not much knowledge about how to measure or ensure the quality properties [85]. Because of this, it is hard to prove the correctness of WAs.

Correctness is an important aspect in providing good-quality WAs. The quality properties are separated in two sections: external and internal qualities. Correctness is one of the external qualities and is part of “Web Content” properties’ category.

If we are referring to the functionality part, we can state that a WA is correct if it behaves according to its specification [69] (based on the requirements, we can decide if the information available in a Web page is valid or not [3]). The first step in creating a correct WA is to make a rigorous analysis to precisely state and analyze the similarities and differences among the various devices and browsers. The development of a WA is an entire process, which after the analysis phase continues with stating a list of precisely formulated properties and then creating the WA (ground) models to hold the properties. The implementation can be described as refinements of those initial abstract models. From this, we could build a way to certify the properties of a WA, by checking the obtained model against the given properties. We can classify the correctness properties into two interesting groups[27]:

- **Correctness properties for session and state management.** The session state should not be affected when, for example, the user navigates away from the page and afterward he or she returns. Replicated parts of the state should be consistent, equivalent between client side and server side, and the state should persist when the client changes (e.g., from desktop to mobile).
- **Application correctness properties.** These properties deal with the dependence of the WA intended behavior on the programming and execution infrastructure (e.g., browser, connection, plug-ins). In order to achieve a precise analysis, a rigorous high-level description is mandatory.

From the fact that no strict way of proving correctness and completeness of the requirements/design exists results a problem regarding the transition from informal to mathematical representations. In [122], the use of ASM ground models (defined as an analysis of dynamic properties of a system using pseudo-code-like descriptions over abstract data structures) is recommended. A characteristic of ground models is the direct correspondence between the interpretation of the system requirements to be modeled and their abstract state machine representation, which simplifies the transition's problem (mentioned previously).

By building an abstract model from the requirements, which satisfies the **CoCoCo-properties**—consistency, correctness, and completeness [23]—we can check whether the WA satisfies the requirements [21]. To fulfill these properties, it means to directly solve the following problems: communication, verification, and validation. Using ASMs for ground models, we can satisfy the properties mentioned before. The simplicity and generality of the ASM language solves the communication problem. The verification problem can be solved by applying standard (pseudo-code) inspection and reasoning. The validation of ASM models can be realized by simulating the ASM runs using the existent tools [25] (e.g., ASM Workbench [40], .NET-executable AsmL engine [16]). CoreASM¹² and ASMETA

¹²<http://sourceforge.net/projects/coreasm/>.

(ASM mETAmodeling)¹³ are two examples of the tools allowing different forms of model analysis for the ASM macros mentioned in the ground models.

In order to reason about correctness of programs and increase their quality, formal verification techniques can be used. By applying model checking to ASMs, one could assure the correctness and quality of software specifications. Farahbod et al. [53] specifies how ASM specifications written in CoreASM can be automatically transformed into Promela specifications, which, afterward, can be verified using the SPIN model checker. AsmetaSMV[13] is a tool that automatically translates ASM specifications written in AsmetaL[55] (the textual notation for ASM models in ASMETA) into models of the NuSMV model checker, and so it allows the verification of computation tree logic (CTL) and linear temporal logic (LTL) formulae.

8.2 *Correctness with Respect to Adaptivity*

Adaptivity to different devices and user preferences is essential for developing a correct WA. A clear condition for a correct WA says that the application should behave corresponding to the specification. If a WA is not adaptive, then it might not function or display correctly on some devices, which means that it would not reflect the specification. The vast number of types of devices and their browsers are big issues for the content and presentation adaptation. The formal model has to describe the particularities of each device and how to change the content and the presentation correspondingly. This could be done by using the client- and server-side adaptation techniques, which require to query the device for its features.

For each of the two main content adaptation techniques which were mentioned before, server-side and content-side adaptation, there already exist frameworks or third-party tools that could be used during the development of an application. The server-side adaptation is realized with the help of the device detection databases, which are available both in the open-source format (e.g., OpenDDR [86]) and in the commercial format (e.g., WURFL [100], DeviceAtlas [1]). A significant difference between the open source and the commercial solutions is how often the databases are being updated. Commercial databases are more reliable, because most of them are daily updated. This is an important issue to check when a device detection database is considered, because an out-of-date database could result in the delivery of erroneous data to the devices. For the client-side adaptation, the “Modernizr” JavaScript framework [14] could be used. The problem of missing browser functionalities could be solved by using replacement code done in JavaScript, the so-called polyfills. Small visual layouts could be also solved on the client side. A big advantage of the server-side adaptation is the loading time, everything that loads on the server will load faster [48]. When significant changes have to be done, the

¹³<http://asmeta.sourceforge.net/>.

server offers a high level of control in fine-tuning. Both of the abovementioned techniques are presenting limitations in discovering all the properties of a device. On client side, the physical nature of the device cannot be determined (e.g., OS version, model, maximum HTML size), but on server side, it is not possible to determine the real-time information (e.g., GPS coordinates, device orientation). On server side, the user could cause problems in detecting the device by changing the user agent (UA). The UA is a string available in the HTML header of a Web page and it is used to query the device detection databases. A similar problem could happen on client side, even without the intervention of the user—many browsers return false positives for certain query tests.

9 Related Work

It is beyond the scope of this chapter to discuss the vast literature of formal modeling mobile systems and *service-oriented architectures (SOAs)*, but we refer to some surveys on these fields [32, 37, 96].

However, if we would like to put our work in context, we should mention first of all [110], in which one of the first examples is presented for representing various kinds of published services as a pool of resources, like it is in our model.

The application of the concept of mobile ambients in the development of distributed SOAs is not a novel idea. One of the first research works that investigated and analyzed location-based services whose availability is related to the surrounding physical environment of the user is [73]. This work has not considered mobility yet, but it introduces the notion of service domains. These domains refer to geographical boundaries which are associated with a set of services that is available for the user within the boundary. In other words, Loke et al. [73] describes a new kind of location-based service discovery architecture.

In the software development community, a UML-based modeling approach called *service-oriented architecture modeling language (SoaML)* [102] has started to become increasingly popular for modeling service-oriented architectures. Moreover, SoaML has an extension called Ambient-SoaML [4, 6] which combines SoaML with the concept of mobile ambients for modeling service-oriented mobile applications.

Another modeling technique is *cloud modeling language (CloudML)* [33, 42], which aims at facilitating the specification of provisioning, deployment, monitoring, and adaptation concerns of multi-cloud systems at design time. Furthermore, CloudML also contains the models@run-time environment for enacting the provisioning, deployment, and adaptation of these systems, as well as for monitoring their status at run time.

Ambient-PRISMA [5] is an aspect-oriented software architectural approach for modeling and developing distributed and mobile applications, which is also extended with the ambient concept. Ambients appear in the (UML-like) meta-model of PRISMA as some special kind of connectors that model the notion of

location and offer mobility services to the components. Mobility of architectural elements is supported by reconfiguring the software architecture. Although SoaML, Ambient-SoaML, CloudML, and Ambient-PRISMA are advanced model-driven engineering techniques, which have more or less been integrated with the software development practice, they are definitely not formal methods and they have no relation to any mathematically rigorous formal specification and verification techniques unlike our approach.

Another research similar to ours is *Cloud Calculus* [67], which is built upon ambient calculus for capturing the dynamic topology of cloud computing systems. Cloud Calculus is effective to verify whether global security policies are preserved after virtual machine migrations, but it is a very specific tool which is not applicable for giving the formal specification of functionalities of cloud/distributed systems.

There exists also some research works which apply ambient logic [35, 39] tailored for ambient calculus to formally verify various security protocols (e.g., authentication and key agreement protocol (AKA) [125]). Furthermore, [41] presents a general algorithm on how the processes expressed by ambient calculus can be model checked against formulas of the ambient logic.

In [26], the ambient concept (notion of “nestable” environments where computation can happen) is introduced into the ASM method, such that the definition of ambient ASM is based upon the semantics of ASM without any changes. But ambient ASM, on which our model is based, is not the only research which aims to build in a concept of mobile ambients to the ASM method. In [113], some advantages of a simple ambient concept introduced into ASM are demonstrated. Although this work was also inspired by ambient calculus, it is by far not as refined and versatile as ambient ASM.

Our approach according to which service instances must be always equipped with unique operations such that they compose the interface of a service instance was originally applied among others in the definition of *Abstract State Services* (AS^2s). AS^2s were introduced first in [75] and were extended and described in detail in [76, 77]. The theory of AS^2s integrates a customized ASM thesis for database transformations [98] as well. In an AS^2 , there are views on some hidden database layer that are equipped with service operations denoted by unique identifiers. The definition of AS^2s also includes the *pure data services* (service operations are just database queries) and the *pure functional services* (operation without underlying database layer) as extreme cases.

For an algebraic formalization of plots, *Kleene algebras with tests* ($KATs$) [70] have been applied in [78]. Prior to this work, the formalization of algebraic plots was founded in [94, 95]. In [18], the idea of ASM-based plot expressions was outlined. Then in [97], it was described in detail how KAT expressions in plots can be replaced with *assignment free* ASMs, which have more expressive power.

In [78], a formal high-level specification of service cloud is given. This work is similar to ours in some aspects. Namely, it applies the language-independent AS^2s with algebraic plots for representing services. But it principally focuses on service specification, service discovery, service composition, and orchestration of service-

based processes; and it does not apply any formal approach to define either static or dynamically changing structures of distributed system components.

Workflow-based solutions have often been applied in case of Web service orchestration for controlling execution of some combination of activities. The *Business Process Execution Language (BPEL)* [68, 74] is a typical representative of this approach, where a workflow can be specified such that the given Web services can be run sequentially, in parallel or even iterated. But in contrast to algebraic plot-based solutions equipped with the concept of service operations, services appear as indivisible components without being interleaved in a BPEL orchestration.

With respect to identity management, there have been several attempts to define a client-centric approach [115]. The author of the paper [2] describes a privacy-enhanced user-centric identity management system allowing users to select their credentials when responding to authentication requests. It introduces “*a category-based privacy preference management for user-centric identity management*” using a CardSpace compatible selector for Java and extended privacy utility functions for P3PLite and PREP languages. The advantage of such a system is that it allows users to select the specific attributes that will eventually be sent to a relying party. Such a system works well for enhancing privacy; however, it fails to address the extra overhead inflicted on the user. As the paper [92] shows, a typical user would tend to ignore obvious security and privacy indicators. For composite services, the authors of the paper [124] describe a universal identity management model focused on anonymous credentials. The model “*provides the delegation of anonymous credentials and combines identity meta-system to support easy-to-use, consistent experience and transparent security.*”

From a client-centric perspective, Microsoft introduced an identity management framework (CardSpace) aimed at reducing the reliance on passwords for Internet user authentication while improving the privacy of information. The identity meta-system, introduced with Windows Vista and Internet Explorer 7, makes use of an “*open*” XML-based framework allowing portability to other browsers via customized plug-ins. However, CardSpace does suffer from some known privacy and security issues, mentioned in the papers [8, 87]. The concept of a client-centric identity meta-system is thoroughly defined in the paper [36]. The framework proposed here is used for the protection of privacy and the avoidance of unnecessary propagation of identity information while at the same time facilitating exchange of specific information needed by Internet systems to personalize and control access to services. By defining abstract services, the framework facilitates the interoperation of the different meta-system components.

Passwords managers can, in our opinion, reside within the topic of automatic authentication for individual users. Projects such as KeePass[90] and LastPass [72] do offer similar functionalities to the IdMM. However, both fall short in two key criteria. Neither of them is truly automatic, since some input is required by the user upon authentication to a service, and while both work well with individual users, they cannot be adapted for SMEs.

The content adaptation topic presented major interest for the mobile device direction. Different solutions for cross-platform mobile development are available,

like cross-platform compilers/applications and building HTML5 or HTML5 hybrid applications [44]. Native mobile applications are not applicable to our problem, because we do not want to create a corresponding application for every mobile operating system; our scope is developing a general application which respects an ASM specification. Cremin [47] explains shortly the different mobile Web content adaptation techniques. Each technique has its own advantages and disadvantages; one should choose the technique that better suits the project, after analyzing the requirements. In our case, the hybrid approach suits the best, so we want to achieve something similar to what [91] presents. Still, our solution would like to use the device detection database as few times as possible, only when the information regarding the properties cannot be retrieved on the client side. The content of a Web page is adapted on the server side corresponding to the properties detected on the client side. Another technique is responsive design, which is combining the Cascading Style Sheets (CSS) media queries with the flexible images and is using the flexible grid technique to scale the page [47]. An example of how to use responsive design for creating a mobile application is presented in [62]. Regardless of the adaptation technique used, the previous works do not apply a formal method in order to prove the correctness of their system.

10 Conclusions

In this chapter among others, we described a high-level formal model of a cloud service architecture in terms of a novel formal approach. The applied method is able to incorporate the major advantages of the ASMs and of ambient calculus. Namely, by this, one is capable to specify in the same formal model of a distributed system both the long-range mobility via several boundaries in a dynamically changing spatial hierarchy and the algorithms of executable components. Since this cloud model is the first nontrivial model which is based on this method, our work also revealed how viable is in practice our two abstraction layers formal approach.

Our cloud model applies a new client-cloud interaction solution based on algebraic plots by which service owners are able to fully control the usages of their services in the case of each subscription, respectively.

The ASM formal models we presented can allow model analysis at early stages of system design, by applying different validation techniques on them, like simulation or scenario construction, through the use of one of the existing ASM tools (e.g., CoreASM, ASMETA). This, together with the verification (model checking of properties) of our models, is part of our future work.

Client-To-Client Interaction Besides the cloud service architecture model, we also discussed a high-level formal definitions of some novel client-to-client interaction features, by which not only information but cloud service functions can be also shared among the cloud users. Our approach is general enough to manage a

situation in which a shared version of a cloud service is shared again several times by several users.

Furthermore, if we shift the client-to-client functionality to client side and wrap into a middleware as it is proposed in Sect. 3 and depicted on Fig. 2b, then no traces of the user activities belonging to the shared services will be left on the cloud. The reason for this is because all the service operations which are shared via a channel are used on behalf of its initial distributor. This consideration can lead one step into the direction of anonym usage of cloud services. The consequence of this is that if a cloud user who has contracts with some service providers completely or partially shares some services via a channel, then he or she should be aware of the fact that all generated costs caused by the usage of these shared services will be allocated to him or her.

Identity Management Machine The current specification of the IdMM, presented in Sect. 6, is thus far limited to the IdMM_{Core} , IdMM_{Client} , IdMM_{User} , and IdMM_{Cloud} agents, allowing for an automatic authentication tool for cloud-based services in the direct and obfuscated interaction scenarios. Since the specification of the $\text{IdMM}_{Protocol}$ and $\text{IdMM}_{Provisioning}$ is currently an ongoing task, we intend to study the various open protocols used in identity management (such as LDAP) or in authentication and authorization (such as OpenID and OAuth). With this study complete, we will then refine the cloud-based functions regarding the authentication via the formally mentioned protocols. Our plan is to first allow IdMM to use external identity providers to authenticate to services while IdMM only takes care of automating the process. With this complete, we can then specify the $\text{IdMM}_{Protocol}$ agent which will act as a stand-alone identity provider communicating with the client's directory via the IdMM_{Client} agent.

With the specification of the $\text{IdMM}_{Protocol}$ agent complete, we can then detail the $\text{IdMM}_{Provisioning}$ agent which will allow for an easier management of the identities and access rights stored both on the client and the cloud provider's systems. The agent will be responsible for the creation, modification, and deletion of identities on the client's directory as well as account creation, synchronization, and deletion on the necessary cloud services. The system will also be responsible for periodical passwords resets on cloud services.

The final step in our research is to apply an access management component to the IdMM. The specification of the IdMM core agent allows for the enforcement of access rights via the rules for the *AuthorizeLogin* state. Further research in the realm of access rights management is required before a full specification of the access management component is achieved.

Cloud Content Adaptivity In Sects. 7.2 and 7.3, the readers can discover how a content adaptation system, created for the interaction between the client's devices and the cloud, can be represented using ASM ground models and how these models are refined to reach implementation phases. We identified four different agents, which are executing the algorithms defined by the corresponding ground models. These models are designed using the ASM formal modeling method, which gives us the possibility of validating and verifying the system.

Further work includes the refinement of all ASM ground models presented in Sect. 7.2, as we did in Sect. 7.3, which means that we will go on with writing the ASM macros, refine them, and finally reach the development phase. The refinement of the ASM macros will lead to the system prototype's implementation. Future research could include the specification of the adaptation rules for each HTML element corresponding to different categories. It is not mandatory to have a different rule for every device; we could also define a rule per group/category. Kumar and Kumar [71] and Yang et al. [123] are also dealing with HTML adaptation, by using the DOM content extraction and rule repositories, respectively, by adopting description logics for the creation of a hierarchical ontology. In our case, the rules will be specified in terms of ASMs.

Acknowledgements This research has been supported by the Christian Doppler Society.

References

1. Afilias Technologies Ltd: Mobile device detection solution—deviceatlas. <https://deviceatlas.com/> (2013)
2. Ahn, G.J., Ko, M., Shehab, M.: Privacy-enhanced user-centric identity management. In: IEEE International Conference on Communications, 2009. ICC '09, pp. 1–5 (2009). doi:[10.1109/ICC.2009.5199363](https://doi.org/10.1109/ICC.2009.5199363)
3. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modeling methods for web application verification and testing: state of the art. *Softw. Test. Verif. Reliab.* **19**(4), 265–296 (2009). doi:[10.1002/stvr.v19:4](https://doi.org/10.1002/stvr.v19:4). <http://dx.doi.org/10.1002/stvr.v19:4>
4. Ali, N., Babar, M.: Modeling service oriented architectures of mobile applications by extending soaml with ambients. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009. SEAA '09, pp. 442–449 (2009). doi:[10.1109/SEAA.2009.25](https://doi.org/10.1109/SEAA.2009.25)
5. Ali, N., Ramos, I., Solis, C.: Ambient-prisma: ambients in mobile aspect-oriented software architecture. *J. Syst. Softw.* **83**(6), 937–958 (2010). doi:<http://dx.doi.org/10.1016/j.jss.2009.12.009>. <http://www.sciencedirect.com/science/article/pii/S0164121209003161> [Software Architecture and Mobility]
6. Ali, N., Chen, F., Solis, C.: Modeling support for mobile ambients in service oriented architecture. In: IEEE First International Conference on Mobile Services (MS), 2012, pp. 1–8 (2012). doi:[10.1109/MobServ.2012.18](https://doi.org/10.1109/MobServ.2012.18)
7. Alpár, G., Hoepman, J.H., Siljee, J.: The identity crisis, security, privacy and usability issues in identity management. *CoRR abs/1101.0427* (2011)
8. Alrodhan, W., Mitchell, C.: Addressing privacy issues in cardspace. In: Third International Symposium on Information Assurance and Security, 2007. IAS 2007, pp. 285–291 (2007). doi:[10.1109/IAS.2007.12](https://doi.org/10.1109/IAS.2007.12)
9. Altenhofen, M., Börger, E., Lemcke, J.: An abstract model for process mediation. In: Proceedings of the 7th International Conference on Formal Methods and Software Engineering, ICFEM'05, pp. 81–95. Springer, Berlin/Heidelberg (2005). doi:[10.1007/11576280_7](https://doi.org/10.1007/11576280_7). http://dx.doi.org/10.1007/11576280_7
10. Amazon Web Services: Amazon elastic compute cloud (amazon ec2). <http://aws.amazon.com/ec2/> (2014)
11. Apache Software Foundation: Apache directory. <http://directory.apache.org/apacheds/> (2013)
12. Apache Software Foundation: Apache tomcat. <http://tomcat.apache.org/> (2013)

13. Arcaini, P., Gargantini, A., Riccobene, E.: AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications. In: Proceedings of the 2nd International Conference on Abstract State Machines, Alloy, B and Z (ABZ 2010). Lecture Notes in Computer Science, vol. 5977, pp. 61–74. Springer, Heidelberg (2010)
14. Ateş, F., Irish, P., Sexton, A., Seddon, R., Farkas, A.: Modernizr: the feature detection library for html5/css3. <http://modernizr.com/> (2013)
15. Azure, M.: Azure: Microsoft’s cloud platform. <https://azure.microsoft.com/> (2014)
16. Barnett, M., Schulte, W., Tillmann, N.: Using asml for runtime verification. In: Börger, E., Gargantini, A., Riccobene, E. (eds.) Abstract State Machines 2003. Lecture Notes in Computer Science, vol. 2589, pp. 407–407. Springer, Berlin/Heidelberg (2003). doi:[10.1007/3-540-36498-6_24](https://doi.org/10.1007/3-540-36498-6_24). http://dx.doi.org/10.1007/3-540-36498-6_24
17. Beste, F.: The model prover: a sequent-calculus based modal π -calculus model checker tool for finite control π -calculus agents. Master’s thesis, Department of Computer Science, Uppsala University (1998). <ftp://ftp.docs.uu.se/pub/mwb/x4.ps.gz>
18. Binemann-Zdanowicz, A., Thalheim, B.: Modeling information services on the basis of ASM semantics. In: Proceedings of the Abstract State Machines 10th International Conference on Advances in Theory and Practice, ASM’03, pp. 408–410. Springer, Berlin/Heidelberg (2003). <http://dl.acm.org/citation.cfm?id=1754749.1754777>
19. Blass, A., Gurevich, Y.: Abstract state machines capture parallel algorithms. ACM Trans. Comput. Logic **4**, 578–651 (2003). doi:<http://doi.acm.org/10.1145/937555.937561>. <http://doi.acm.org/10.1145/937555.937561>
20. Blass, A., Gurevich, Y.: Abstract state machines capture parallel algorithms: correction and extension. ACM Trans. Comput. Logic **9**, 19:1–19:32 (2008). doi:<http://doi.acm.org/10.1145/1352582.1352587>. <http://doi.acm.org/10.1145/1352582.1352587>
21. Bolis, F., Gargantini, A., Guarnieri, M., Magri, E., Musto, L.: Model-driven testing for web applications using abstract state machines. In: Grossniklaus, M., Wimmer, M. (eds.) Current Trends in Web Engineering. Lecture Notes in Computer Science, vol. 7703, pp. 71–78. Springer, Berlin/Heidelberg (2012). doi:[10.1007/978-3-642-35623-0_7](https://doi.org/10.1007/978-3-642-35623-0_7). http://dx.doi.org/10.1007/978-3-642-35623-0_7
22. Börger, E.: The asm refinement method. Form. Asp. Comput. **15**(2–3), 237–257 (2003). doi:[10.1007/s00165-003-0012-7](https://doi.org/10.1007/s00165-003-0012-7). <http://dx.doi.org/10.1007/s00165-003-0012-7>
23. Börger, E.: Construction and analysis of ground models and their refinements as a foundation for validating computer based systems. Form. Asp. Comput. **19**(2), 225–241 (2007). doi:[10.1007/s00165-006-0019-y](https://doi.org/10.1007/s00165-006-0019-y). <http://dx.doi.org/10.1007/s00165-006-0019-y>
24. Börger, E., Rosenzweig, D.: The wam—definition and compiler correctness. In: Beierle, C., Plümer, L. (eds.) Logic Programming: Formal Methods and Practical Applications. Studies in Computer Science and Artificial Intelligence, vol. 11, pp. 20–90. North-Holland, Amsterdam (1995)
25. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, New York (2003)
26. Börger, E., Cisternino, A., Gervasi, V.: Ambient abstract state machines with applications. J. Comput. Syst. Sci. (Special Issue in honor of Amir Pnueli) **78**(3), 939–959 (2012). doi:[10.1016/j.jcss.2011.08.004](https://doi.org/10.1016/j.jcss.2011.08.004). <http://dx.doi.org/10.1016/j.jcss.2011.08.004>
27. Börger, E., Cisternino, A., Gervasi, V.: Contribution to a rigorous analysis of web application frameworks. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) Integrated Formal Methods. Lecture Notes in Computer Science, vol. 7321, pp. 1–20. Springer, Berlin/Heidelberg (2012). doi:[10.1007/978-3-642-30729-4_1](https://doi.org/10.1007/978-3-642-30729-4_1). http://dx.doi.org/10.1007/978-3-642-30729-4_1
28. Bósa, K.: A formal model of a cloud service architecture in terms of ambient asm. Technical report, Christian Doppler Laboratory for Client-Centric Cloud Computing (CDCC), Johannes Kepler University Linz, Hagenberg (2012)

29. Bósa, K.: An ambient asm model for client-to-client interaction via cloud computing. In: Proceedings of the 8th International Conference on Software and Data Technologies (ICSOFT), Reykjavik, Iceland, pp. 459–470. SciTePress (2013). doi:[10.5220/0004490904590470](https://doi.org/10.5220/0004490904590470). <http://www.icsoft.org/>. (Best Paper Award)
30. Bósa, K.: An Ambient ASM Model for Cloud Architectures. *Acta Cybernetica* (2014). Submitted
31. Bósa, K.: Formal modeling of mobile computing systems based on ambient abstract state machines. *Semant. Data Knowl. Bases* **7693**, 18–49 (2013). doi:[10.1007/978-3-642-36008-4_2](https://doi.org/10.1007/978-3-642-36008-4_2). http://dx.doi.org/10.1007/978-3-642-36008-4_2
32. Boudol, G., Castellani, I., Hennessy, M., Kiehn, A.: A theory of processes with localities. *Form. Asp. Comput.* **6**, 165–200 (1994). doi:[10.1007/BF01221098](https://doi.org/10.1007/BF01221098). <http://dx.doi.org/10.1007/BF01221098>
33. Brandtzaeg, E., Parastoo, M., Mosser, S.: Towards a domain-specific language to deploy applications in the clouds. In: *Cloud Computing 2012: 3rd International Conference on Cloud Computing, Grids, and Virtualization*, pp. 213–218. IARIA (2012)
34. Brunette, G., Mogull, R.: Security guidance for critical areas of focus in cloud computing V2. 1. <http://goo.gl/PxAeP> (2009)
35. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). *Inf. Comput.* **186**(2), 194–235 (2003)
36. Cameron, K., Posch, R., Rannenber, K.: Proposal for a common identity framework: a user-centric identity metasystem. <http://goo.gl/3q2sF> (2008)
37. Cardelli, L.: Mobility and security. In: Bauer, F.L., Steinbrüggen, R. (eds.) *Foundations of Secure Computation Proceedings of the NATO Advanced Study Institute. Lecture Notes for Marktoberdorf Summer School 1999 (A summary of several Ambient Calculus papers)*, pp. 3–37. IOS Press, Amsterdam (1999)
38. Cardelli, L., Gordon, A.D.: Mobile ambients. *Theor. Comput. Sci.* **240**(1), 177–213 (2000)
39. Cardelli, L., Gordon, A.D.: Anytime, anywhere: modal logics for mobile ambients. In: *POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 365–377. ACM (2000)
40. Castillo, G.: The asm workbench: a tool environment for computer-aided analysis and validation of abstract state machine models. In: Margaria, T., Yi, W. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science*, vol. 2031, pp. 578–581. Springer, Berlin/Heidelberg (2001). doi:[10.1007/3-540-45319-9_40](https://doi.org/10.1007/3-540-45319-9_40). http://dx.doi.org/10.1007/3-540-45319-9_40
41. Charatonik, W., Gordon, A., Talbot, J.M.: Finite-control mobile ambients. In: Métayer, D.L. (ed.) *Programming Languages and Systems. Lecture Notes in Computer Science*, vol. 2305, pp. 295–313. Springer, Berlin/Heidelberg (2002). doi:[10.1007/3-540-45927-8_21](https://doi.org/10.1007/3-540-45927-8_21). http://dx.doi.org/10.1007/3-540-45927-8_21
42. Chauvel, F., Ferry, N., Morin, B., Rossini, A., Solberg, A.: Models@Runtime to support the iterative and continuous design of autonomous reasoners. In: Bencomo, N., France, R., Götz, S., Rumpe, B. (eds.) *MRT 2013: 8th International Workshop on Models@run.time at MODELS 2013: ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems. CEUR Workshop Proceedings* (2013)
43. Chelemen, R.M.: Modeling a web application for cloud content adaptation with asms. In: *International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, 2013, pp. 44–51 (2013). doi:[10.1109/CLOUDCOM-ASIA.2013.76](https://doi.org/10.1109/CLOUDCOM-ASIA.2013.76)
44. Chipperfield, R.: An introduction to cross-platform mobile development technologies. <http://www.codeproject.com/Articles/388811/An-introduction-to-cross-platform-mobile-developme> (2012)
45. Christian-Albrechts-Universität zu Kiel: Visual programming of databases - visual sql. <http://www.informatik.uni-kiel.de/en/is/miscellaneous/visualsql> (2008)
46. Cloud Security Alliance: Top threats to cloud computing. <http://goo.gl/wLd7m> (2010)
47. Cremin, R.: Mobile web content adaptation techniques. <http://mobiforge.com/starting/story/mobile-web-content-adaptation-techniques> (2011)

48. Cremin, R., Passani, L.: Server-side device detection: history, benefits and how-to. <http://mobile.smashingmagazine.com/2012/09/24/server-side-device-detection-history-benefits-how-to/> (2012)
49. Dam, M.: Model checking mobile processes. In: Best, E. (ed.) CONCUR'93, 4th International Conference on Concurrency Theory. Lecture Notes in Computer Science, vol. 715, pp. 22–36. Swedish Institute of Computer Science/Springer, Kista/Berlin/Heidelberg (1993). Full version in Research Report R94:01
50. Dhamija, R., Dussault, L.: The seven flaws of identity management: usability and security challenges. *IEEE Secur. Priv.* **6**(2), 24–29 (2008). doi:[10.1109/MSP.2008.49](https://doi.org/10.1109/MSP.2008.49)
51. Dold, A.: A formal representation of abstract state machines using pvs. Technical report, University Ulm (1998)
52. ENISA: Cloud computing. benefits, risks and recommendations for information security. Technical report, The European Network and Information Security Agency. <http://www.enisa.europa.eu/activities/risk-management/files/deliverables/cloud-computing-risk-assessment> (2009)
53. Farahbod, R., Glässer, U., Ma, G.: Model checking coreasm specifications. In: Proceedings of the 14th International Abstract State Machines Workshop (ASM'07) (2007)
54. Gargantini, A., Riccobene, E.: Encoding abstract state machines in pvs. In: Gurevich, Y., Kutter, P., Odersky, M., Thiele, L. (eds.) *Abstract State Machines: Theory and Applications*. Lecture Notes in Computer Science, vol. 1912, pp. 303–322. Springer, Heidelberg (2000)
55. Gargantini, A., Riccobene, E., Scandurra, P.: A metamodel-based language and a simulation engine for abstract state machines. *J. Univers. Comput. Sci.* **14**(12), 1949–1983 (2008)
56. Gervasi, V.: An asm model of concurrency in a web browser. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) *Abstract State Machines, Alloy, B, VDM, and Z*. Lecture Notes in Computer Science, vol. 7316, pp. 79–93. Springer, Berlin/Heidelberg (2012). doi:[10.1007/978-3-642-30885-7_6](https://doi.org/10.1007/978-3-642-30885-7_6). http://dx.doi.org/10.1007/978-3-642-30885-7_6
57. Google: Google apps for business. <http://www.google.com/enterprise/apps/business/> (2014)
58. Google Chrome: What are extensions?. <http://developer.chrome.com/extensions/index.html> (2013)
59. Google Developers: Google web toolkit. <https://developers.google.com/web-toolkit/> (2013)
60. Google Developers: Google app engine: platform as a service . <https://developers.google.com/appengine/> (2014)
61. Gordon, A.D., Cardelli, L.: Equational properties of mobile ambients. *Math. Struct. Comp. Sci.* **13**, 371–408 (2003). doi:[10.1017/S0960129502003742](https://doi.org/10.1017/S0960129502003742). <http://dl.acm.org/citation.cfm?id=966815.966816>
62. Grigsby, J.: Responsive design for apps. <http://blog.cloudfour.com/responsive-design-for-apps-part-1/> (2013)
63. Gunjan, K., Sahoo, G., Tiwari, R.K.: Identity management in cloud computing - a review. *Int. J. Bus. Forecast. Market. Intell.* **1**(4) (2012). <http://www.ijert.org>
64. Gurevich, Y.: Evolving algebra 1993: Lipari guide. In: *International Conference on Functional Programming*, pp. 9–36. Oxford University Press, New York (1994)
65. Gurevich, Y.: Sequential abstract state machines capture sequential algorithms. *ACM Trans. Comput. Logic* **1**, 77–111 (2000). doi:[http://doi.acm.org/10.1145/343369.343384](https://doi.org/10.1145/343369.343384). <http://doi.acm.org/10.1145/343369.343384>
66. Jaakkola, H., Thalheim, B.: Visual SQL – high-quality ER-based query treatment. In: Jeusfeld, M.A., Pastor, O. (eds.) *Conceptual modeling for novel application domains*. In: *Proceedings of ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM*, Chicago, IL, 13 October 2003. Lecture Notes in Computer Science, vol. 2814, pp. 129–139. Springer, Heidelberg (2003)
67. Jarraya, Y., Eghtesadi, A., Debbabi, M., Zhang, Y., Pourzandi, M.: Cloud calculus: security verification in elastic cloud computing platform. In: Smari, W.W., Fox, G.C. (eds.) *CTS*, pp. 447–454. IEEE, Denver (2012). <http://dblp.uni-trier.de/db/conf/cts/cts2012.html#JarrayaEDZP12>

68. Juric, M.B.: Business Process Execution Language for Web Services BPEL and BPEL4WS, 2nd edn. Packt Publishing, Birmingham (2006)
69. Kappel, G.: Chapter I: Web applications. University Lecture. <http://is.uni-paderborn.de/fileadmin/Informatik/AG-Engels/Lehre/WS1213/WE/slides/WE-2012-01.pdf> (2012)
70. Kozen, D.: Kleene algebra with tests. *Trans. Program. Lang. Syst.* **19**(3), 427–443 (1997)
71. Kumar, V., Kumar, A.: Client device based content adaptation using rule base. *J. Comput. Sci.* **7**(12), 1908–1913 (2011)
72. LastPass: Lastpass—the last password you have to remember. <https://lastpass.com/> (2013)
73. Loke, S.W., Krishnaswamy, S., Naing, T.T.: Service domains for ambient services: concept and experimentation. *Mob. Netw. Appl.* **10**(4), 395–404 (2005). <http://dl.acm.org/citation.cfm?id=1160162.1160165>
74. Louridas, P.: Orchestrating web services with BPEL. *IEEE Softw.* **25**(2), 85–87 (2008). <http://doi.ieeecomputersociety.org/10.1109/MS.2008.42>
75. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: Abstract state services. In: Object-Oriented and Entity-Relationship Modelling/International Conference on Conceptual Modeling/The Entity Relationship Approach, pp. 406–415 (2008). doi:10.1007/978-3-540-87991-6_48
76. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: Composing personalised services on top of abstract state services. In: Delcambre, L., Kaschek, R.H., Mayr, H.C. (eds.) *The Evolution of Conceptual Modeling*, no. 08181 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl (2008). <http://drops.dagstuhl.de/opus/volltexte/2008/1597>
77. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: A theory of data-intensive software services. *Serv. Orient. Comput. Appl.* **3**(4), 263–283 (2009)
78. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: A formal model for the interoperability of service clouds. *Service Oriented Computing and Applications* **6**(3), 189–205 (2012). doi:10.1007/s11761-012-0101-7. <http://cdcc.faw.jku.at/publications/kdschewe/schewe2012FMCloud.pdf>
79. Mather, T., Kumaraswamy, S., Latif, S.: *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, Inc., Sebastopol (2009)
80. Mell, P., Grance, T.: The nist definition of cloud computing. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> (2011)
81. Microsoft: Office 365. <http://office.microsoft.com/en-us/> (2014)
82. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, parts I and II. *Inform. Comput.* **100**(1), 1–77 (1992). doi:10.1016/0890-5401(92)90008-4. [http://dx.doi.org/10.1016/0890-5401\(92\)90008-4](http://dx.doi.org/10.1016/0890-5401(92)90008-4)
83. Nida, P., Dhiman, H., Hussain, S.: A survey on identity and access management in cloud computing. *Int. J. Eng. Res. Technol.* **3**(4) (2014). <http://www.ijert.org/>
84. Novell: Ldap classes for java. http://www.novell.com/developer/ndk/ldap_classes_for_java.html (2013)
85. Offutt, J.: Web software applications quality attributes. In: *Quality Engineering in Software Technology (CONQUEST 2002)*, pp. 187–198 (2002). <http://www.cs.gmu.edu/~offutt/rsrch/papers/conquest02.pdf>
86. OpenDDR LLC: Openddr - the best open and completely free device description repository with access apis available worldwide. <http://www.openddr.org/> (2013)
87. Oppliger, R., Gajek, S., Hauser, R.: Security of microsoft's identity metasytem and cardspace. In: *ITG-GI Conference Communication in Distributed Systems (KiVS)*, pp. 1–12 (2007)
88. Prodromou, E.: Openid privacy concerns. <http://goo.gl/WIDYx> (2007)
89. Pusch, C.: Verification of compiler correctness for the wam. In: von Wright, J., Grundy, J., Harrison, J. (eds.) *Theorem Proving in Higher Order Logics (TPHOLs'96)*. Lecture Notes in Computer Science, vol. 1125, pp. 347–362. Springer, Berlin (1996)
90. Reichl, D.: Keepass password safe. <http://www.keepass.info/contact.html> (2013)

91. Reiger, B., Rieger, S.: Adaptation: why responsive design actually begins on the server. <http://www.slideshare.net/yiibu/adaptation-why-responsive-design-actually-begins-on-the-server> (2012)
92. Schechter, S., Dhamija, R., Ozment, A., Fischer, I.: The emperor's new security indicators. In: IEEE Symposium on Security and Privacy, 2007. SP '07, pp. 51–65 (2007). doi:10.1109/SP.2007.35
93. Schellhorn, G.: Verifikation Abstrakter Zustandsmaschinen. Ph.D. thesis, University Ulm (1999)
94. Schewe, K.D., Thalheim, B.: Reasoning about web information systems using story algebras. In: Advances in Databases and Information Systems, ADBIS, pp. 54–66 (2004)
95. Schewe, K.D., Thalheim, B.: Conceptual modelling of web information systems. *Data Knowl. Eng.* **54**(2), 147–188 (2005)
96. Schewe, K.D., Thalheim, B.: Personalisation of web information systems: a term rewriting approach. *Data Knowl. Eng.* **62**(1), 101–117 (2007). doi:DOI:10.1016/j.datak.2006.07.007. <http://www.sciencedirect.com/science/article/pii/S0169023X06001406>
97. Schewe, K.D., Thalheim, B.: Term rewriting for web information systems: termination and Church-Rosser property. In: Proceedings of the 8th International Conference on Web Information Systems Engineering, WISE'07, pp. 261–272. Springer, Berlin/Heidelberg (2007). <http://dl.acm.org/citation.cfm?id=1781374.1781404>
98. Schewe, K.D., Wang, Q.: A customised ASM thesis for database transformations. *Acta Cybern.* **19**(4), 765–805 (2010). <http://dl.acm.org/citation.cfm?id=1945572.1945579>
99. Schewe, K.D., Bosa, K., Lampesberger, H., Ma, J., Rady, M., Vleju, B.: Challenges in cloud computing. *Scal. Comput. Pract. Exp.* **12**(4), 385–390 (2011)
100. ScientiaMobile, Inc: Wurfl - mobile device database by scientiamobile. <http://wurfl.sourceforge.net/> (2013)
101. Sermersheim, J.: Lightweight directory access protocol (ldap): the protocol. RFC. <http://tools.ietf.org/html/rfc4511> (2006)
102. Service Oriented Architecture Modeling Language (SoaML): Specification for the UML Profile and Metamodel for Services (UPMS) Revised Submission. OMG document: ptc/2009-04-01 (2009)
103. Shaarawy, M.: Cloudification of visual sql. Master's thesis, Johannes Kepler University Linz (2013)
104. Song, H., Compton, K.J.: Verifying π -calculus processes by promela translation. Technical report, Department of Electrical Engineering and Computer Science University of Michigan, Ann Arbor (2003)
105. Spielmann, M.: Abstract state machines: verification problems and complexity. Ph.D. thesis, RWTH Aachen (2000)
106. Spielmann, M.: Model checking abstract state machines and beyond. In: Proceedings of the International Workshop on Abstract State Machines, Theory and Applications, ASM '00, pp. 323–340. Springer, London (2000). <http://dl.acm.org/citation.cfm?id=647752.734545>
107. Stärk, R.F., Nanchen, S.: A logic for abstract state machines. *J. Univers. Comput. Sci.* **7**(11), 980–1005 (2001)
108. Stärk, R.F., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer, Heidelberg (2001)
109. Sturru, E.: Identity and access management in a cloud computing environment. Master's thesis, Econometric Institute, Erasmus School of Economics, Erasmus University Rotterdam (2011). http://thesis.eur.nl/pub/10422/MA-5%20IENE%20Sturru_294763.pdf
110. Tanaka, Y.: Meme Media and Meme Market Architectures: Knowledge Media for Editing, Distributing, and Managing Intellectual Resources. Wiley, New York (2003). <http://books.google.at/books?id=tezm83Wiqv8C>
111. Thalheim, B.: Visual SQL: towards ER-based object-relational database querying. In: Proceedings of the 27th International Conference on Conceptual Modeling, ER '08, pp. 520–521. Springer, Berlin/Heidelberg (2008). doi:10.1007/978-3-540-87877-3_41. http://dx.doi.org/10.1007/978-3-540-87877-3_41

112. The Open Group Identity Management Work Area: Identity management. <http://goo.gl/ssPTu> (2004)
113. Valente, M., Bigonha, R., Loureiro, A., Maia, M.: Abstractions for mobile computation in ASM. In: Graham, P., Maheswaran, M. (eds.) Proceedings of the International Conference on Internet Computing, IC 2000, 26–29 June, pp. 165–172. CSREA Press, Las Vegas (2000)
114. Venters, W., Whitley, E.A.: A critical review of cloud computing: researching desires and realities. *J. Inf. Tech.* **27**(3), 179–197 (2012)
115. Vleju, M.B.: A client-centric asm-based approach to identity management in cloud computing. In: Advances in Conceptual Modeling. Lecture Notes in Computer Science, vol. 7518, pp. 34–43. Springer, Berlin/Heidelberg (2012). doi:[10.1007/978-3-642-33999-8_5](https://doi.org/10.1007/978-3-642-33999-8_5). http://dx.doi.org/10.1007/978-3-642-33999-8_5
116. Vleju, M.B.: A client-centric identity management tool for small and medium enterprises using cloud services. In: 4th Workshop on Software Services, pp. 15–19. Bled, Slovenia (2012). <http://www.cloudconference.eu/>
117. Vleju, M.B.: Interaction of the idmm with a client-side identity management component. Technical report, Christian Doppler Laboratory for Client-Centric Cloud Computing (CDCC), Johannes Kepler University Linz, Hagenberg (2012)
118. Vleju, M.B.: IdMM demo. <http://youtu.be/DoM36D0ydkA> (2013)
119. Vleju, M.B.: A practical implementation of a client-centric identity management tool for cloud computing. In: EUROCAST-Computer Aided Systems Theory, Gran Canaria (2013)
120. Vleju, M.B.: Automatic authentication to cloud-based services. *J. Univers. Comput. Sci.* **20**(3), 385–405 (2014). http://www.jucs.org/jucs_20_3/automatic_authentication_to_cloud
121. Winter, K.: Model checking for abstract state machines. Ph.D. thesis, Technical University of Berlin (2001)
122. Yaghoubi Shahir, H., Farahbod, R., Glässer, U.: Refactoring abstract state machine models. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) Abstract State Machines, Alloy, B, VDM, and Z. Lecture Notes in Computer Science, vol. 7316, pp. 345–348. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30885-7_28](https://doi.org/10.1007/978-3-642-30885-7_28). http://dx.doi.org/10.1007/978-3-642-30885-7_28
123. Yang, S.H., Zhang, J., Huang, A., Tsai, J.P., Yu, P.: A context-driven content adaptation planner for improving mobile internet accessibility. In: IEEE International Conference on Web Services, 2008. ICWS '08, pp. 88–95 (2008). doi:[10.1109/ICWS.2008.31](https://doi.org/10.1109/ICWS.2008.31). <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4670163>
124. Zhang, Y., Chen, J.L.: Universal identity management model based on anonymous credentials. In: IEEE International Conference on Services Computing (SCC), pp. 305–312 (2010). doi:[10.1109/SCC.2010.46](https://doi.org/10.1109/SCC.2010.46)
125. Zhang, X., Li, X., Luo, W.: Aka protocol and its formal analysis and verification using ambient calculus and logics. In: International Conference on Networking and Digital Society, vol. 1, pp. 194–197 (2009). doi:[http://doi.ieeecomputersociety.org/10.1109/ICNDS.2009.54](https://doi.org/10.1109/ICNDS.2009.54)

W*H: The Conceptual Model for Services

Ajantha Dahanayake and Bernhard Thalheim

Abstract Services as an emerging paradigm in modern information technology (IT) infrastructures underwent the first hype for service-oriented computing caused by Web services and the second hype by IT market pressures on large corporations (e.g. SAP), leading to standardisations incorporating logical-level specifications leaving much of the low-level details unaccounted for.

The conception of a service needs a conceptual reflection. However, the service notation lacks a conceptual model. This gap is caused by the variety of aspects that must be reflected, such as the handling of the services as a collection of offerings, a proper annotation facility beyond ontologies, a tool to describe the service concept and the specification of the added value of a business user. Those requirements must be handled at the same time. Therefore, this chapter contributes to the development of a conceptual model of a service through a specification framework W*H and through an embedding framework to the concept-content-annotation triptych and Hermagoras of Temnos inquiry frames.

1 Introduction

The promise of a service as a design artifact for innovation, design and evolution in the information systems domain has not yet lived up to expectations. It is still a concept debated for gauging its value in terms of usefulness, usage and usability as an IT artifact. Research has looked into the service mainly from two perspectives, (a) from the low-level technological point of view and (b) from the higher abstract business point of view. Unfortunately, both perspectives add up to the confusion leaving unanswered: what is really a service description model, where does it belong

A. Dahanayake (✉)

Department of Computer Information Science, Prince Sultan University, Riyadh, Kingdom of Saudi Arabia

e-mail: adahanayake@pscw.psu.edu.sa

B. Thalheim

Department of Computer Science, Christian Albrechts University Kiel, 24098 Kiel, Germany

e-mail: thalheim@is.informatik.uni-kiel.de

© Springer International Publishing Switzerland 2015

B. Thalheim et al. (eds.), *Correct Software in Web Applications and Web Services*,

Texts & Monographs in Symbolic Computation,

DOI 10.1007/978-3-319-17112-8_5

to and what is the main purpose or added value of following a service-centred design approach?

In order to close this gap between main service design initiatives and their abstraction level interpretations, we propose an inquiry-based conceptual model for service systems designing. The W*H model closes the gap created so far by technology- and business-level service models. It streamlines the communication through an inquiry-based service systems modelling approach. The W*H model bases its foundation on an inquiry structure that provides the designer with a model to formulate right questions for completeness, simplicity and correctness into service systems innovation, design and development. We introduce the W*H model as the conceptual model for service systems conception, designing and innovation.

1.1 The Service Concept

The field of information systems (IS) has shifted towards advanced and cross-disciplinary IT systems design and development referred to as service systems engineering [26]. For example, medical, environmental, disaster recovery, life science, etc., are such domains that largely invest on advanced cross-disciplinary systems embracing the notion of service to enhance situation-specific needs through IT (service) systems.

Following this trend, more and more organisations define, develop and deploy cross-disciplinary service systems, making those resulting applications available for end users to combine those into end-user situational services in ways that the developers may have not originally planned or intended [12]. The implication of this shift is that those service systems are then subject to evaluations of systems functioning based on its trustworthiness, flexibility to change and efficient manageability and maintainability. Therefore, today's systems development endeavours demand their developers to understand the systems functioning within its domain of usage and its service [6]. The service has gained ground and recognition as the more realistic concept for dealing with complexities of cross-disciplinary systems engineering extending beyond the classical information systems development realm.

The first hype for service-oriented computing has been caused by Web services. This led to standardisation efforts that targeted the logical level of specification. It has been realised that the huge manifold in preferred approaches led to a mixture of inconsistent, incomprehensible and not integrate-able approaches [12]. The second hype for service-oriented computing came with the pressure from the IT market, especially for large corporations such as SAP. Again, specification is done at the logical level with many low-level details that cannot be incorporated [12].

Service science is an initiative of IBM [14] that launched the emerging academic field for studying service systems to discover underlying principles that can guide the innovation, design and development of service systems. As a distinct interdisciplinary field, it searches for an ideology and a unifying paradigm [27].

1.2 Levels of Abstractions

The service is being defined using different abstraction models with varying applications representing a multitude of definitions of the service concept [10]. The increasing interests in services have introduced service concept's abstraction into levels such as business services, Web services, software as a service (SaaS), platform as a service (Paas) and infrastructure as a service (Iaas) [4].

Service architectures are proposed as means to methodically structure systems [3, 7, 28]. The service delivery discussions in research have mainly concentrated on the relationship between an economic activity of increasing the business value and technology as a means, in this case, Web services to deliver effective and efficient services. As a consequence, the economic activity that supports a business process is defined as a business service and the Web service design, development and orchestration as the identification of the right services and organisation of a manageable hierarchy of composite services for choreographing in supporting a business process [20].

However, the service delivery has paid no attention to the innovation, design and development of a service as an IT artifact [10]. The researches that have used the concept of service in IT artifact innovation, design and development named it an information service [28] and have moved on to the innovation, design and development of a conceptual model of an IT artifact taking a standard design and development point of view and not a service-centred conceptual model.

Although services are developed, used, applied and intensively discussed in practice, the service concept does not have a conceptual model or a description that guides the innovation, design and development of a service. Furthermore, the relevance of a service to the organisational environment must be explicitly specified [6].

1.3 IT Service Systems

There is a substantial subset of service systems that can be described as “information intensive”, and it is desirable to take a more abstract view of service contexts that highlight what person-to-person, self-service and automated or computational services have in common rather than emphasising their differences [28]. Service systems combine and integrate the value created in different design contexts like person-to-person encounters, technology-enabled self-service, computational services and multichannel, multi-device and location-based and context-aware services [27].

The IT service system view reveals the intrinsic design challenges that derive from the nature of the information required to perform a service and emphasises the design choices that allocate the responsibility to provide this information between the service provider and service consumer. Taken together, the information

requirements and the division of labour for satisfying them determine the nature and intensity of the interactions in the service system. This more abstract approach that applies to all contexts overcomes many of the limitations of design approaches that focus more narrowly on the distinctive concerns of each context [9].

1.4 Survey on the Chapter

This chapter aims at a general notion for a conceptual model for services: A general definition of a service seems to be rather difficult. We use however a framework that separates concerns such as service as a product, service as an offer, service request, service delivery, service application, service record, service log or archive and service exception. This separation of concern allows supporting a general characterisation of services by their ends, their stakeholders, their application domain, their purpose and their context.

The chapter is organised as follows: Sect. 2 revisits the related works highlighting the uniqueness of our contribution. Section 3 introduces our model approach to services. Section 4 discusses the reflections and grounding of the conceptual model and conceptualisation using the concept-content-annotation triptych and embedding the added value into the service description. In Sect. 5, we summarise the classical rhetoric frame introduced by Hermagoras of Temnos that generalises some of the concepts found in the resource-event-agent (REA) framework. In this manner, we are able to *separate* conceptual models for service specification from conceptual models for service systems and both from the IT system realisation. Typically, these aspects are mixed. Section 6 provides an evaluation and discussion on W*H model, and Sect. 7 summarises the conclusions and future research issues. Finally, in the Appendix, we illustrate the W*H specification for a medical service system.

2 Service Models and Modelling Approaches

2.1 The REA (Resource-Event-Agent) Ontology

REA ontology's conceptual origin lies in the traditional accounting applications which use the double-entry bookkeeping technique for managing financial systems where business transactions are recorded as a credit and a debit, thus a double entry. REA ontology formulated as in the original article [18] was further articulated and extended by others, e.g. [13]. The core concepts in the REA ontology are resources, economic event and agent. The fundamental behind the ontology is that there are two ways agents can increase or decrease the value of their resources: through exchange and conversion process [13].

An economic resource is a valuable good, right or service that at a given point is under the identifiable control of an economic agent. An economic resource is under the control of an economic agent if that person owns the resources or otherwise is able to derive economic benefit from it. If two economic agents desire to obtain control over one or more economic resources controlled by the other agent, then both agents may wish to engage as trading partners in an economic exchange, which is a business transaction that transfers the control of resources between agents. A transfer of control of a resource(s) from one agent to another agent is modelled as an economic event in which the concerned resources are identified as stockflow relation and agents anticipate in provider and receiver roles. Economic reciprocity in exchanges is modelled through the duality relation between economic events and requesting events such as payments, in which the provider and receiver roles of the involved agents are switched.

2.2 The RSS Model

The Resource-Service-Systems (RSS) model for service systems [21] is an adaptation of the REA model, stressing that REA is a conceptual model of economic exchange. It is not a model of service exchange, because of the influence of service-dominant logic (SDL) [34] in the RSS. SDL has been proposed as the philosophical foundation of service science for providing the right perspective, vocabulary and assumptions to build a theory of service science, their configurations and models of interaction [34]. SDL sees all economic activities as service exchanges between service systems [35]. In SDL, service is a competence that exchanges for the benefit of other service systems. In contrast to the traditional goods-dominant logic (GDL) model, SDL sees a service as a collaborative process in which each party brings in or makes accessible its unique resources. The RSS model serves as a generalised model for positioning service within the resources and systems, but it does not help in conceptualising a service at the event of innovation, design and development of a service as an IT artifact.

2.3 The Model of the Three Perspectives of Services

The three perspectives of services—abstraction, restriction and cocreation—were introduced as a conceptual model of service concept that views services as perspectives on the use and offering of resources [4]. The perspectives addressed by this conceptual model are service as a means for abstraction, service as means for providing restricted access to resources and service as a means for cocreation of value. It relies on the argument that in the classical manufacture economic model, service has been defined and characterised by identifying properties such as intangibility, inseparability, heterogeneity and perishability in the goods-dominant

logic (GDL) model [35] but that there are other kinds of resources that cannot be distinguished from those that have been identified in GDL and are seen as problematic for services. It has been suggested to stop searching for properties of services that uniquely define them and instead to view and investigate services as perspectives on the use and offering of resources. This model too has its origin of adaptation and extension in the REA model and can be categorised as a generalised model for the service concept that comes short of identifying the distinctive nature of service as an IT artifact for the service systems innovation, design and development.

2.4 Web Service Description Languages

Much of scientific research in service systems are being dominated by Web service modelling and conceptualisation of structural and behaviour dependencies of Web services [8]. These Web service modelling initiatives, e.g. [8], are seeking full-fledged modelling languages for providing the appropriate conceptual model for developing and describing Web services and their composition [19, 22, 36]. The Web service domain concentrates on service-oriented architectures (SOAs), software systems decomposed into independent units and named services that interact with one another through message exchanges. The main goal is to promote reuse and evolve ability, as they start at early phases as possible, describing these interactions in the development life cycle. From standards such as Business Process Execution Language (BPEL) [19] and Web Service Choreography Description Language (WS-CDL) [36] to languages with purposes derived from their requirements [23] overshadow the service systems and service concept in contrary to our motivation of a conceptual model for a service.

In the reflections of SOA, Organization for the Advancement of Structured Information Standards (OASIS), it is evident that services are a combination of a technical and a social concept [17] and that most of the desired expectations in the use of SOA-based systems are rooted in social rather than of physical ones. The creation of value in the context of service-dominant logic in contrast to goods-dominant logic is also paramount.

2.5 The Seven Contexts of Service Design

A description of the contexts that combine person-to-person encounters, technology-enhanced encounters, self-service, computational services and multichannel, multi-device and location-based and context-aware services description is presented in [17, 27]. The characteristic concerns and methods of those seven different design contexts are represented as a unifying view spanning over the information-intensive service systems design and modelling paradigm [9]. The focus is on the information required to perform the service, how the responsibility to provide this

information is divided between the service provider and service consumer and the patterns that govern information exchange yielding a more abstract description of service encounters and outcomes. Thereby, it makes it easier to see the systematic relationships among the contexts that can be exploited as design parameters or patterns, such as the substitutability of stored or contextual information for person-to-person interactions.

This view of the seven contexts for service design [9] reveals the intrinsic design challenges that are inherent within the nature of the information required to perform a service and emphasises the need of design choices that allocate the responsibility to provide this information between the service provider and service consumer. The information requirements and the division of labour for satisfying them determine the nature and intensity of the interactions in the service system.

A service concept for the service design research has been identified in [10, 14, 27] as the key concept for service innovation, design and development. It describes a service concept as a how and what a service design. It is used as a mediate between customer needs and organisation strategic intents. While the service concept is widely used, very little has been said in terms of service innovation, design and development. In order to overcome many of the limitations of design approaches that focus more narrowly on the distinctive concerns of each context, a service description language with a more abstract approach that applies to all contexts is required.

A service needs to integrate the social, physical and technological aspects in order to provide a service that creates value with social effects. Therefore, in the following sections, a general notion for a conceptual model for the service is outlined and defined as a framework that separates concerns such as service as a product, service as an offer, service request, service delivery, service application, service record, service log or archive and also service exception, which allows supporting a general characterisation of services by their ends, their stakeholders, their application domain, their purpose and their context.

3 Models and Services

3.1 *The Notion of the Model*

We use the notion of the model in [32]. Any artifact can be considered to be a model. IT services are, however, specific artifacts. These specifics must be reflected in the notion of a service model. We introduce a general notion of a model and then specialise this notion to IT services.

An artifact \mathcal{A}^* is implicitly based on its *background* consisting of a *basis* (e.g. paradigms, postulates, restrictions, culture, conventions, common sense), a *grounding* (e.g. concepts, foundations), some *context* and a *community of practice*.

An artifact is called well formed if it satisfies a well-formedness criterion. It has a profile. The profile is based on the goal or purpose or function of the artifact. An artifact can be analogous to another artifact \mathcal{A} based on some analogy criterion. It can be simpler or more focused than \mathcal{A} based on some complexity or focus criterion. We call an artifact \mathcal{A}^* *adequate* for \mathcal{A} if it is well formed, purposeful for the profile, simpler than \mathcal{A} and analogous to \mathcal{A} .

An artifact is *justified* by a justification, i.e. [11] by empirical corroboration (according to purpose, background, etc.) for the representation of an artifact \mathcal{A} , by rational coherence and conformity, by falsifiability and by stability and plasticity. The artifact is *sufficient* by its *quality* characterisation for internal quality, external quality and quality in use. The artifact \mathcal{A}^* is called *dependable* if it is justified and sufficient.

An artifact \mathcal{A}^* is called *model* of \mathcal{A} if it is *adequate* and *dependable*. A model \mathcal{A}^* has its background and is going to be used within it.

The model and the artifact are *functioning* if there are methods for the utilisation of the artifact in dependence on the profile of the artifact. Functioning artifacts have their capability and capacity. Artifacts are used for application cases. These cases are embedded into stories of model application such as description-prescription, explanation, optimisation-variation, verification-validation-testing, reflection-optimisation, exploration, hypothesis development, documentation-visualisation or also substitution. We call an artifact and a model *effective* if it can be deployed according to its portfolio.

3.2 *The Purpose of a Service Model*

IT services are artifacts. The typical purposes of a service model can be:

- (a) Communication about services: The model allows to communicate among stakeholders about existing and missing properties of the artifact, about integration of the artifact into other artifacts, about the quality that the artifact has in certain application scenarios, about functional and effective properties, etc. The model describes the artifacts that are under consideration.
- (b) Construction of a service: Services may also be developed. In this case, the model serves as a blueprint and documentation for their development. The model prescribes the service. It is an archetype for the construction.
- (c) Realisation of a service: The model can also be used as a kind of a contract during realisation. The artifact that is going to be developed conforms to the model. The model is an antetype for the realisation. This conformance is verifiable.

The third purpose is often considered to be the main purpose for IT systems. The second and especially the first purpose are often neglected. In this chapter, we concentrate on the second and especially on the first purposes. Purposes such as control of an artifact, substitution of another artifact within a third one, upgrade or maintenance are not in the centre of consideration in this chapter. We may however extend this paper into such directions.

Let us consider the communication purpose of a service model. A model is used by one stakeholder for communicating his/her understanding of the world to another stakeholder. The success of this communication depends on these stakeholders, especially their abilities to deliver and to understand the model. Each of them has their background, their cultures, their knowledge, their experience and their contexts. Therefore, the model pragmatics (especially “phonetics”) and the model pragmatism (especially “rhetoric”) influence the added value. Pragmatics is dependent on the presentation, appearance and form (“gestalt”) of the model.

Communication is segmented into communication acts. The success of communication depends on the quality of the model, the relationship among the stakeholders, their background and context, their abilities and experience and the “layout” of the model. This relationship in a model communication story is depicted in Fig. 1. The goals, purposes and functions of A and B may be different. A model function for stakeholder A is determined by its function in the description scenario of the artifact by an adequate and dependable model. A model function of stakeholder B is its function in the validation scenario of the model by comparing the model to the artifact represented by the model.

The construction model for a virtual service can be considered similar to software construction. The model serves as a blueprint or prototype for the construction of a service. Construction of services is the art, science and craft of changing a vision into reality through a model to operate something that fulfils a human need. Construction of services typically uses several models. These models are bound to each other and represent different aspects of the service.

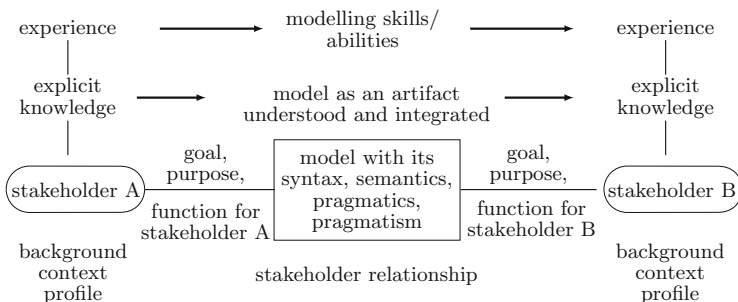


Fig. 1 Dimensions of communicating by models

3.3 *The Background, the Community of Practice and the Context of a Service Model*

Services are dependent on culture, habit, behaviour, personality and common behaviour. The background of any artifact is the complex of physical, cultural and psychological factors that serves as the environment of a service. It is the set of conditions against which an occurrence is perceived. It consists of the *basis* from one side, i.e. paradigms, postulates, restrictions, theories, culture, conventions, common sense and *grounding*, to the other side, i.e. concepts, foundations, language as carrier and cargo.

The background of an artifact is typically implicit. It is, however, an essential element of the meta-description of the artifact. It allows to understand how an artifact can be used, how it can be integrated into other artifacts and how it should not be used. A manufactured artifact is typically made on the basis of techniques, commonly accepted approaches and a general understanding of the application domain. Paradigms contain the basic assumptions, ways of thinking and methodology that are commonly accepted by the stakeholders in the application domain. Postulates or fundamental principles are something that are taken as self-evident or assumed without proof as a basis for understanding. Conceptions can be combined into a theory.

The foundation is the groundwork of the artifact. Concepts are used for classification. Classification depends on the context and deployment. Concepts are all the knowledge that the person has and associates with the concept's name. Narrative artifacts use a language for their expression and presentation. The cargo defines what is transferred or represented via a model. It is anything put in or on something for conveyance depending on the purpose.

The context surrounds the artifact and helps to determine its deployment. Context reflects general characterisations, categorisation, utilisation and general descriptions such as quality. Elements of a context of a service are the application domain or the discipline, the school of thought, time, space, granularity and scope. For services we may distinguish between the infrastructure context (e.g. IT systems, communication and exchange support), provider context, developer context, usage context (based on the simple rhetoric frame who-where-what-when-why), context of the service user, temporal context with different notions of time (e.g. introduction, applicability, validity, availability and stakeholder-determined time) and the organisational and social context.

The background of a service model provides the basis for faith into the service. The context of a service evaluates the utility of the service. A service is accepted by a community of practice depending on their roles, e.g. developers, users, supporters and competitors. Criteria for acceptance are learnability, ergonomics, consistency, observance of standards, feedback and robustness and ethnography of user environments. Services must be coherent with user expectations and intuition, must accommodate a wide range of stakeholder skills and should arrange judgements

consistent with its importance. The service must be perceptible and uses in an appropriate way the space and resources (parsimonious).

We may now summarise the discussed properties of a service model by *PEST*: paradigmatic for the basis, environment aware, reflecting the society (and scientific) culture and technologically founded.

3.4 Adequacy for Service Models

Adequacy of models is given if the model is well formed, simple, analogue and purposeful. Adequacy of service models refines these properties. Well-formed models of services must be trackable, i.e. provide deadlines, limits and benchmarks and states when, how long, when to terminate and to interrupt and what to do first. Simplicity is based on an explicit consideration of the purpose and an explicit specification how this purpose can be achieved by whom, what, when, where, which and why. Vagueness is not permitted. Analogy of service models can be specified by meaningfulness and by acceptability. Meaningfulness measures the progress of the service against the vision based on an evaluation how much, how many and when it is accomplished. Purposefulness of service models is given if the service is realistic, is doable and provides a substantial progress; whether the service is worthwhile, can be provided at the right time and matches efforts and needs of the community of practice; and whether it is acceptable for correction and evolution.

This refinement of adequacy for IT services is similar to criteria applied in engineering. A service model must thus be *SMART* (simple, meaningful, acceptable, realistic and trackable).

3.5 Dependability for Service Models

Dependability of models is defined through justification and quality evaluation. This characterisation can now also be refined for services. We need a notion of corroboration, coherence and conformity, falsifiability and stability and plasticity. Corroboration of a service is straightforward: a service must have an added value. Therefore, the service is described by its potential benefits and values. It is positively stated. The service value must be understood by their users. This understanding is essential before an agreement for deployment of a service can be reached. Understandability supports coherence, conformity and falsifiability. At the same time, a service must be rewarding. We put the purpose of a service into a wider context of driving. The rewarding property defines the plasticity and stability of the service.

Evaluation of services can be based on quality measures developed in software engineering. We distinguish between internal quality, external quality and quality of use. These corresponding characteristics of these measures may thus also be applied for the evaluation of services.

We observe, therefore, that a service must be *PURE* (positively stated, understandable, rewarding and evaluated).

3.6 *Functioning and Effective Service Models*

Models are used in dependence on the existence of corresponding methods for their utilisation. They are functioning if they provide the necessary capability and are effective in their deployment. The capability level can be described similar to the SPICE [16] as follows:

1. *Performing and executing*: The purposes of the model are satisfied.
2. *Managing and defining*: The model can be successfully applied depending on the purpose.
3. *Establishing and controlling*: The model is adequate and dependable, is well documented and allows to understand its added value and potential.
4. *Understanding, predicting and performing with sense*: The background and context of the model are well understood by the community of practice. The model can be applied due to the power of the methods.
5. *Optimising*: The application of the model is well supported and improves the situation in an application compared with the situation without the existence of the model.

The usefulness and usability characterisation of functioning and effectiveness of the service model can be characterised depending on the capability level. A model should function. Functioning of a service model can be characterised by *CLEAR* (challenging, legal, environmentally sound, appropriate and recorded): The service model has to keep the motivation high without being unrealistic. Methods should be legal. Their consequence should be known. The appropriateness of a service model is supported by the fitness to experience and skills and by unlocking the potential in an application. Finally, the deployment of the model should be recordable with an explicit log of the impact.

Models are used in usage or deployment scenarios. This deployment defines the goals and purposes of models. Such models are then used as instruments in processes such as defining, constructing, exploring, communicating, understanding, replacing, documenting, negotiating, reporting and accounting. Users of models deploy them within their environment for their goals.

The usage of models can be characterised by *SCOPE* (situation, core competencies, obstacles, prospects and expectations). The service models are characterised in the same way. The situation determines the impact, actions and external and internal factors. The model capability is based on the strengths, abilities and facilities,

foundation and the added value. These characteristics define the core competencies of a model. The obstacles of a model are given by the potential issues and threats. At the same time, the prospects of a model are given by possibilities and experiences, chances and opportunities both internally and externally. Finally, a model must meet the expectations, i.e. the future-view anticipated developments in external and internal conditions that could influence their impact.

4 The Descriptive Framework for Service Models

The PEST-SMART-PURE-CLEAR-SCOPE framework for models of IT services combines many different facets of a model. We thus have to develop an aspect-oriented descriptive framework. This descriptive framework may be based on dimensions of services.

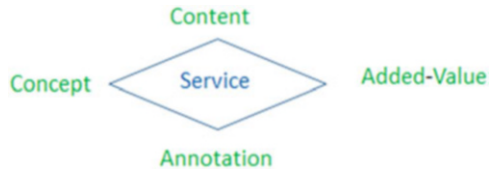
While the service-oriented computing hypes repeated the logical-level specification with inadequate support for many low-level details, it also repeatedly confirmed the need of an aid for semantics and pragmatics with respect to service specification descriptions and a conceptual model.

4.1 *Dimensions of Services*

The conception of a service needs the reflections on conceptualisation. In essence, there is a need to consider a conceptual model and a description of a service specification framework with syntactics, semantics and pragmatics, which constitute the semiotics [31]. It is necessary to concentrate on the conceptualisation of content for a given context considering annotations with respect to organisation intentions, motivations, profiles and tasks; thus, we need at the same time sophisticated annotation facilities far beyond ontologies. We also need a tool for description of the concept of a service. The description of scenarios inside a certain organisational environment and preliminary considerations of general service offering guarantee reliable quality-based content coordinated to consumer groups [29].

Finally, we must specify the added value of a service for a business user. These requirements must be handled at the same time leading to the development of a conceptual model of a service through a specification framework. Then the organisation's business services are composed of the content space, the concept space, the annotation space [29] and the added value space. For this reason, we consider a service to be consisting of content, concepts, annotation and added value. These dimensions are interdependent from each other (Fig. 2).

Fig. 2 Conceptualisation of service as orthogonal dimensions of concept, content, annotation and added value



4.2 *The Content Dimension: Services as a Collection of Offerings*

The service defines the what, how and who on what basis of service innovation, design and development and helps mediate between customer or consumer needs and an organisation's strategic intent [14]; when extended above the generalised business and technological abstraction levels, the content of the service concept composes the need to serve the following purposes [12]:

- Fundamental elements for developing applications
- Organising the discrete functions contained in (business) applications comprised of underlying business process or workflows into interoperable (standards-based) services
- Services abstracted from implementations representing natural fundamental building blocks that can synchronise the functional requirements and IT implementation perspective
- Services to be combined, evolved and/or reused quickly to meet business needs. Represent an abstraction level independent of underlying technology

The abstraction of the notion of a service system within an organisation's strategic intent emphasised by those purposes given above allows us to define the content description of services by a collection of offers that are given by companies, by vendors, by people and by automatic software tools[5]. Thus, the content of a service system is a collection of service offerings. Figure 3 specifies the content dimension.

The service offering reflects the supporting means in terms of with what means the service's content represented in the application domain. It corresponds to the identification and specification of the problem within an application area. The problem is a specific application case that resides with an organisational unit. Those problems are subject to events that produce triggers needing attention. Those triggering events have an enormous importance for service descriptions. They couple to the solution at hand that is associated with how and what IT solutions are required.

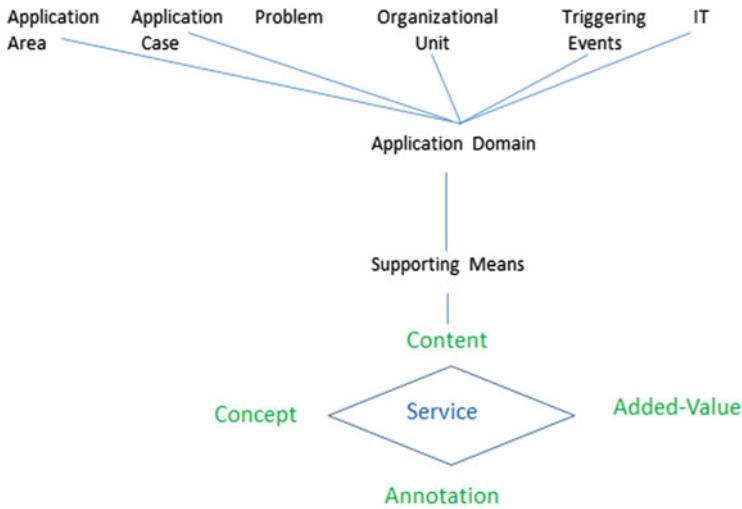


Fig. 3 The content dimension of services

4.3 The Annotation Dimension

According to [24], annotation with respect to arbitrary ontologies implies general purpose reasoning supported by the system. Their reasoning approaches suffer from high computational complexities. As a solution for dealing with high worst-case complexities, the solution recommends a small-size input data. Unfortunately, it is contracting the impossibility of ontologies and defining content as complex-structured macro data. It is therefore necessary to concentrate on the conceptualisation of content for a given context considering annotations with respect to organisation intentions, motivations, profiles and tasks; thus, we need at the same time sophisticated annotation facilities far beyond ontologies. Annotation thus must link the stakeholders or parties involved to the activities and the sources to the content and concept.

4.4 The Parties in Annotation

Parties in services are suppliers, consumers and producers and have their own target, goal, intentions and aims. They are bound by their capabilities, support requirements and information demand. This description constitutes the kernel of the profile. Other components of the profile of a stakeholder in a service process include educational, employment and psychological descriptions. Parties are interested in solving a certain number of tasks. Tasks are ordered and prioritised. A task is an assigned piece of work, which often has to be finished within a certain time by a party or

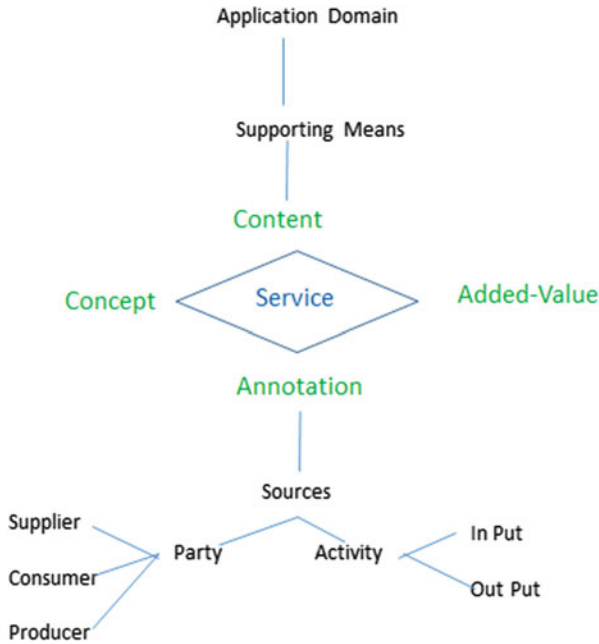


Fig. 4 The annotation dimension of a service

parties whose duty is its completion. It implies work imposed by a user in authority and obligation or responsibility to perform. A task may consist of subtasks, so we assume that tasks can be constructed on the basis of elementary tasks. Figure 4 illustrates the annotation dimension and the parties involved.

The *portfolio* consists of an ordered and prioritised collection of tasks including compensation and is determined by the responsibilities one has and is based on a number of targets. The party portfolio within an application is thus based on a set of tasks an actor has or intends to complete and for which solution the actor has the authority and control, a description of involvement within the task solution and a collaboration that is necessary for solving the task. Tasks are supported by services. A party may activate a service in order to complete a task.

The *involvement* of parties within service activation is based on the specification of the *role* a party plays during execution of the service, the part the party plays within its portfolio and the rights and obligations a party has within the given role. The role specifies the behaviour expected from a party. A role is a comprehensive pattern of behaviour and serves as a strategy for coping with recurrent situations and dealing with the roles of others. A role remains relatively stable, even though different parties occupy the position. A party may have a unique style of role execution, but this is exhibited within the boundaries of the expected behaviour of the party. Role expectations include both actions and qualities. There are two types of roles: declarative and contextual ones. Declarative roles declare that a party is

playing a particular role, e.g. a party being identified as an employee. Contextual roles show how a party acts within the context of an application story and show how a party is involved within the context of another application story. Declarative roles may be modelled by associating the actor to a role type. Contextual roles are modelled by associating an actor with the work effort the actor is assigned to and a role type describing the involvement of the actor. The role type provides a description of the role and can be hierarchically structured. Roles may also be hierarchically structured. At the same time, roles may be played in collaboration.

4.5 The Service Activities in Annotation

Services are input-output activities typically based on a (set of) workflow(s) that must be followed by the party. These services are often virtual goods. A service is *offered* together with policies applicable to the service export. A *service provider* can advertise, modify and withdraw a service. Service providers are often supported by *traders* that store services available from service providers and act as brokers for available services and actual requests. Traders are able to search for the most appropriate service for a given request on the basis of matching criteria and search constraints. They store new advertisements of service providers and categorise them. If the trader has a large variety of services available from providers, then the service offer properties are standardised to service offer property types. A trader has further a trading offer domain and is restricted by trading contexts. The information schema of the traded services enables maintenance of meta-structures for services, categorisation and partitioning. Traders themselves may be organised in trading syndicates or trading communities with trading administrations, internal arbitrators, makers for trading rules and policies, export policy control and trader owners. Trader communities can be federated with external arbitrators.

Services are typically layered depending on the stage a service is used by a party. We distinguish the following stages: First, a service is developed. This stage is similar to a production. Next, a service is quoted. A part may now use the service and request it. The supporting party responds and provides details on the service use. Next, the service is requested. The service is then ordered. Later, the service is delivered. Finally, the service must be billed and paid. Parties involved into this service utilisation process play different roles and parts and have different responsibilities. This service utilisation process is typically layered. It can also be chained or executed partially in parallel or in different chains.

4.6 The Concept of a Service

Conceptual modelling aims at creating an abstract representation of the situation under investigation or, more precisely, the way users think about it. Conceptual models enhance models with concepts that are commonly shared within a community

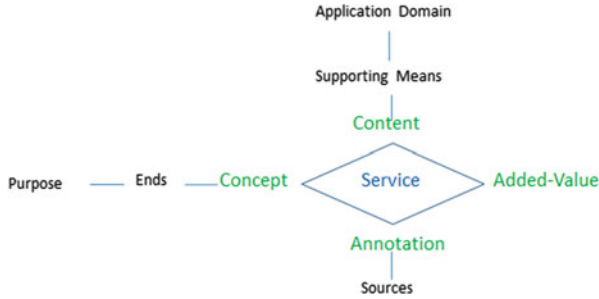


Fig. 5 The concept dimension of a service

or at least between the stakeholders involved in the modelling process. A general definition of concepts is given in [30, 31].

Concepts specify our knowledge on what things are there and what properties things have. Concepts are used in everyday life as a communication vehicle and as a reasoning chunk. Concept definition can be given in a narrative informal form, in a formal way, by reference to some other definitions, etc. We may use a large variety of semantics, e.g. lexical or ontological, logical or reflective.

Conceptualisation aims at collecting concepts that are assumed to exist in some area of interest and the relationships that hold them together. It is thus an abstract, simplified view or description of the world that we wish to represent. Conceptualisation extends the model by a number of concepts that are the basis for an understanding of the model and for the explanation of the model to the user. The definition of the ends or purpose of the service is represented by the concept dimension. It is the curial part that governs the service's characterisation. The purpose defines in which cases a service has a usefulness, usage and usability. They define the potential and the capability of the service.

Figure 5 depicts the *concept* dimension of a service.

4.7 The Added Value Dimension

The added value of a service to a business user or stakeholder is in the definition of surplus value during the service execution. It defines the context in which the service systems exists, the story line associated within the context, which systems must coexist under which context definitions prevailing to time. Surplus value defines the worthiness of the service in terms of time and labour that provide the return of investment (ROI).

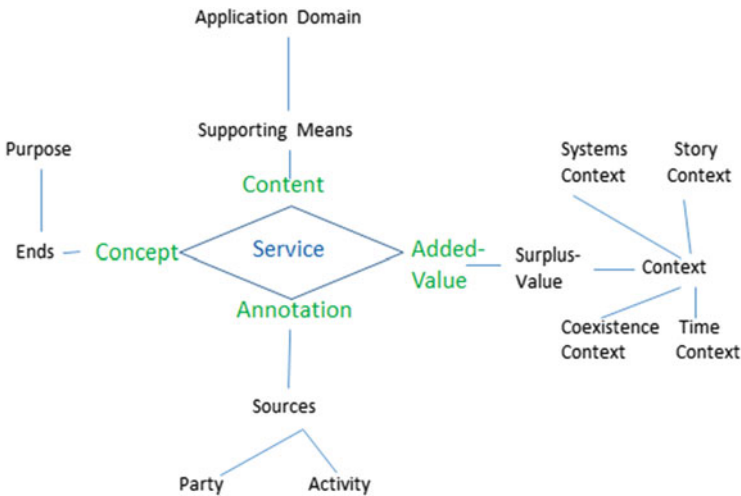


Fig. 6 The added value dimension of a service

Figure 6 illustrates the value and especially the added value dimension of a service.

5 W*H: The Conceptual Model of Services

5.1 Extending the Rhetorical and the Zachman Frameworks

The Zachman framework [37] uses the classical W6H description (who-when-where-what-how-why). The key questions in systems development are:

- Who will be using the system?
- When will the system be used?
- Where is the information system used?
- What is represented in the system?
- How will the system be used?
- Why is the system used?

We observe that there are additional dimensions that are of importance:

- Competency
- Time (schedule, delay)
- Environment (context, technical and organisational)
- Quality (in which quality, with which guarantees)
- Runtime characteristics (adaptation, exceptions, delay)

- Collaboration (with whom, which exchange, on which basis, which portfolio and profile)
- Additional motivation (on which reason)

Additionally, we should take into consideration the policy, intention, goal and aim of the provider. One might ask now whether this list is exhaustive and substantial. Also, we need a prioritisation of these questions. Therefore, we need an approach that allows to consider the main characteristics in a systematic and surveyable way.

We discover that the specification framework may be headed by the questions *who, what, when, where, why, in what way* and *by what means*. This framework has not been developed in the computer age.¹ The success of Zachman's framework as an inquiry system for information systems engineering and Hermagoras of Temnos frames in legal inquiry frameworks inspires the definition stage of our inquiry system for the conceptualisation process.

5.2 *The General Characteristics of Services*

Services are to be characterised by their specific properties, the supplier or manufacturer, the pricing that is applicable and the costs depending on the user, provider and deliverer. Further, services can be kept in an inventory of their providers, their suppliers or their deliverers. Service may be composed of other services. Some information on products is independent of the supplier or provider. Other information, e.g. pricing and availability, depends on the supplier.

This approach is used by the *Service Modelling Language* (SML) by W3C. It becomes very sophisticated with many characteristics that must be given. There is no hierarchy in the specification. An idea we might use for such characterisation is the separation of concern by aspects and layering of specifications depending on the maturity stage. The first approach is the basis for our specification frame that allows us to describe a service. The second approach is used for the specification of an IT service system.

The service is primarily declared by specifying:

- The **ends** or **purpose** (wherefore) of the service and thus the benefit a potential user may obtain when using the service. The **purpose description** governs the service. It allows to characterise the service. This characterisation is based on the answers for the following questions: *why, whereto, for when and for which reason*. We call these properties primary since they define in which cases a

¹ It is far older. It dates back to Cicero and even to Hermagoras of Temnos who was one of the inventors of rhetoric frames in the second century BC. The latter has been using a frame consisting of the seven questions: *quis, quid, quando, ubi, cur, quem ad modum* and *quibus adminiculis* (W7: *who, what, when, where, why, in what way, by what means*). The work of Hermagoras of Temnos is almost lost. He had a great influence on orality due to his proposals. For instance, Cicero has intensively discussed his proposals and made them thus available.

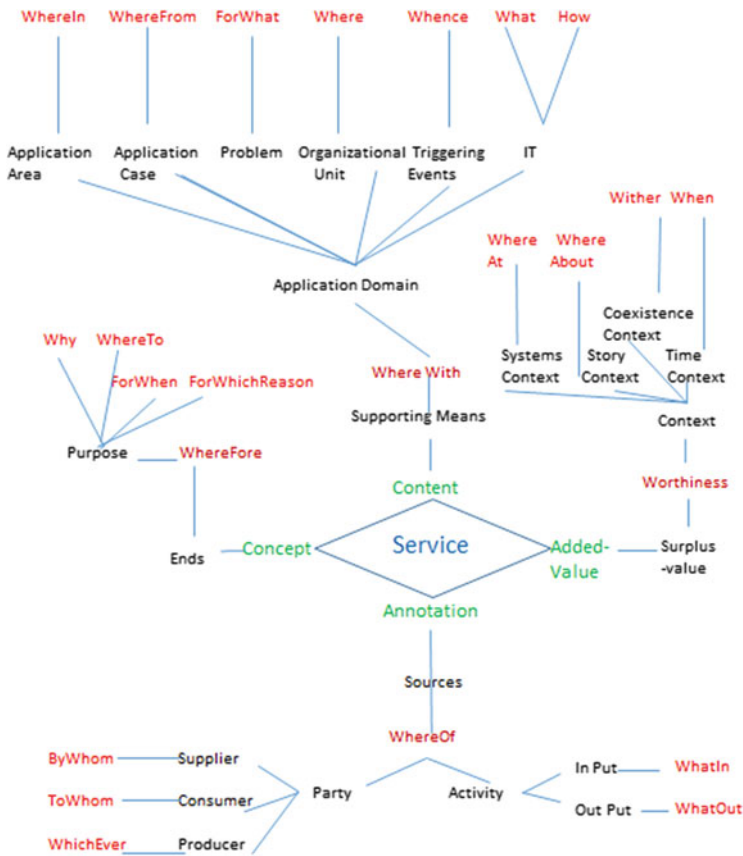


Fig. 7 The W*H inquiry-based conceptual model for services

service has a usefulness, usage and usability. They define the potential and the capability of the service.

- The sources (*whereof*) of the services with a general description of the environment for the service.
- The supporting means (*wherewith*) which must be known to potential users in the case of utilising the service.
- The surplus value (*worthiness*) a service utilisation might give to the user.

Figure 7 combines the service dimensions to a general W*H framework.

A full description of IT services is based on the five-dimensional specification:

- (A) The primary service description describes the service.
- (B) The annotation dimension captures the party and activity dimensions:
 - (a) The party dimension describes the stakeholders involved into a service. Parties may play different roles, may have different parts in the story of

service application, may have obligations and permissions and may also be restricted in their capabilities and competencies. Typical descriptions for the party dimensions are given while answering the *by whom, to whom* and *whichever* questions.

- (b) The **activity dimension** describes the processes played during service application. These processes may use resources, may be supported by functions provided by the service system and, given at the side of the business user of a service, may result in a number of changes to data, to control and to rights. We answer question such as *what in and what out*.
- (C) The **content dimension** captures the supporting means by the application **domain dimension**: describes the problems to be solved by the service, the application area in which a service is usable, the typical approaches within this application area and the typical solutions that are sought for the problems under consideration. We thus answer questions like *wherein, where, for what, where from, whence, what* and *how*.
- (D) The **concept dimension** captures the ends wherefore the purpose of the service answer questions such as *why, whereto, for when* and *for which reason*.
- (E) The **added-value dimension** captures the surplus value that defines the usefulness, usage and usability of the services' worthiness for systems context, coexistence context, story context and time context. Therefore, we might also declare the context characteristics for a service. Context has at least four sides: provider or developer or supplier context for a service, the user context for a service, the system environment context that must exist for service utilisation and the coexistence context for a service within a set of services. Therefore, these context dimensions are declared by answering the following questions: *whereat, whereabouts, whither, and when*.

To summarise, our service description language is thus based on the following questions:

- Primarily: wherefore, whereof, wherewith and worthiness (= W4) and additionally why, whereto, for when and for which reason (= W4)
- Secondly: by whom, to whom, whichever, wherein, where, for what, wherefrom, whence, what and how (= W10H)
- Additionally: whereat, whereabouts, whither and when (= W4)

We may call our framework the **W*H specification framework** where $W^*H = (W4 + W4 + W10H + W4)$ and H stands for how. The kernel of this framework is the $(W4 + W10H)$ questionnaire.

6 W*H : Evaluation and Discussion

Many of the most complex service systems being built and imagined today combine person-to-person encounters, technology-enhanced encounters, self-service, computational services and multichannel, multi-device and location-based and context services. The research reported in this paper has examined characteristic concerns of those different design contexts from a service-centred conceptualisation to propose a unifying view, especially when the service system is an “information-intensive” service system. W*H is a full description of services based on the four-dimensional specification based on primary, secondary and additional questions derived following the work of Hermagoras of Temnos. A focus on the W*H specification framework to perform the service, based on primary, secondary and additional questions, yields a more detailed description of service encounters and outcomes of how the responsibility to provide this information is divided between the service provider and service consumer and the patterns that govern information exchange.

W*H is evaluated by application to a real-life situation—to establish a disease diagnosis decision support network (DDDSN) for ophthalmologists for age-related macular degeneration (ARMD) treatments, a true cross-disciplinary real-life service engineering application (Appendix). This advanced cross-disciplinary DDDSN system has three services for practicing ophthalmologists:

- (1) Decision support image base
- (2) Continuous knowledge enhancement of the image base
- (3) On-demand learning module for specialists and residents to update their diagnosis and knowledge (see Table 1 in the Appendix)

DDDSN initiative exposes the systems designing to the challenges of the evolution of the application domain. This is a crucial challenge for modern cross-disciplinary IT service systems, as per definition, they are Web information systems and are notorious for their low “half-life” period and the high potential of evolution, migration and integration. Therefore, it is necessary to incorporate service modelling into such systems building approaches. To this service engineering challenge, we introduced W*H (Fig. 8), our novel conceptual model for service modelling.

The W*H model in Fig. 8 fulfils the conceptual definition of the service concept composing the need to serve the following purposes:

- The composition of the W*H model consisting of content space, concept space, annotation space and add value space as orthogonal dimensions that capture the fundamental elements for developing applications.
- It reflects number of aspects neglected in other service models, such as the handling of the service as a collection of offering, a proper annotation facility, a model to describe the service concept and the specification of added value. It handles those requirements at the same time.
- It helps capturing and organising the discrete functions contained in (business) applications comprised of underlying business process or workflows into interoperable (standards-based) services.

Service	Service	Name			
Concept	Ends	Wherefore?			
		Purpose	Why?		
			Where to?		
			For When?		
		For Which reason?			
Content	Supporting means	Wherewith?			
		Application Domain	Application are	Wherein?	
			Application case	Wherefrom?	
			Problem	For What?	
			Organizational unit	Where	
			Triggering Event	Whence	
IT	What				
		How			
Annotation	Source	Where of?			
		Party	Supplier	By whom?	
			Consumer	To whom?	
			Producer	Whichever?	
		Activity	In-Put	What in?	
Out-Put	What out?				
Added Value	Surplus Value	Worthiness?			
		Context	Systems Context	Where at?	
			Story Context	Where about?	
			Coexistence Context	Wither?	
		Time Context	When?		

Fig. 8 The W*H service description model

- The model accommodates the services to be abstracted from implementations representing natural fundamental building blocks that can synchronise the functional requirements and IT implementation perspective.
- It considers by definition that the services are to be combined, evolved and/or reused in order to quickly meet business needs.
- Finally, it represents an abstraction level independent of underlying technology.

The W*H model in Fig. 8 fulfils the usefulness, usage and usability requirements for service systems designing composing the needs to serve the following purposes:

- The inquiry through simple and structured questions according to the primary dimension on wherefore, whereof, wherewith and worthiness further leads to secondary and additional questions along the concept, annotation, content, add value or surplus value space that covers usefulness, usage and usability requirements in totality.

- The powerful inquiring questions are a product of the conceptual underpinning of W*H grounded within the conceptual modelling tradition in the concept-content-annotation triptych extended with the added value dimension and further integration and extension with the inquiry system of Hermagoras of Temnos frames.
- The W*H model is comprised of 23 questions in total that cover the complete spectrum of questions addressing the service description: (W4 + W4 + W10H + W4) and H that stands for how.
- The model's compactness helps to validate domain knowledge during solution modelling discussions with the stakeholders with high-demanding work schedules.
- The comprehensibility of the W*H model became the main contributor to the understanding of the domain's services and requirements.
- The model contributes as the primary input model leading to the IT service systems projection on solution modelling.
- It contributes as the primary input model leading to the IT service systems projection on the evaluation criteria of systems functioning on its trustworthiness, flexibility to change and efficient manageability and maintainability.

7 Conclusion

We have made a contribution to one of the main questions that dominate the service systems engineering by presenting W*H model as a consistent, complete and comprehensive conceptual model for service systems engineering. The model notion has been refined to the PEST-SMART-PURE-CLEAR-SCOPE framework for IT services. This framework allows to evaluate whether an artifact is adequate, dependable, functioning and effective within the given context and by its background for a community of practice. The W*H model is extracted from this framework by mapping the properties of the framework to corresponding questions. The question word set is extensible.

The W*H model fills a gap in service models and modelling research arena. It combines and streamlines the discussions between the business abstraction level's business services and technology abstraction level's Web services into an interpretation and terminology valid and understood in both worlds. The W*H model also conceptualises the worthiness into service concept and thereby provides the groundwork and context to evaluate the main purpose and the added value of investing in the implementation of services.

The W*H model in Fig. 8 nevertheless can lead to developing some interesting and ground-breaking future research directions such as:

- In order to follow a service systems development to be mainstream practice, holistic approach for service development is a required future research direction. Most service designers are familiar with some of these contexts, and each context

has a research and practitioner literature that highlights their characteristic design concerns and methods. But few service designers are familiar with all of them, and because the design concerns and methods in one context can seem incompatible with those in others, there is relatively little work that analyses design concerns and methods that span over multiple contexts.

- Correlation and cocreation of business process re-engineering and service systems innovation and engineering is another research direction that has a great potential.
- Another promising research direction is to pursue service engineering as the standard approach for systems quality improvement in business environment through exceptional event detection and event processing and monitoring for business process improvement.
- Research into quantifying the return of investment through service-based engineering which has a potential to explore ROI issues is another research direction that can bring vast advantages to business informatics.
- Another rewarding research direction is in exploring Web information systems: the design, innovation and evolution as part-whole relationship between the existing system and Web information services.

In the Appendix, we demonstrate the application of W*H model for medical services. The framework and the model also provide a basis for a SWOT (strengths, weaknesses, opportunities and threats) analysis of the service.

Appendix: Application of W*H Specification Framework

An ophthalmologic research institute in the Netherlands initiated to establish a disease diagnosis decision support network (DDDSN) for ophthalmologists [1]. The initiative is on the creation of an expert community of ophthalmologists who contribute their knowledge to a repository for validating individual disease diagnoses. The motivation behind this DDDSN proposal is to harvest tangible benefit in the form of shared access to this unique image repository to achieve interoperability and location-independent decision support. The system is for the benefit of specialists all over the world to diagnose diseases at early stages and for treating their patients with up-to-date disease diagnosis decision support.

Medical diagnosis decision support is hardly in existence [25]. An impressive amount of medical images are daily generated in hospitals and medical centres. Consequently, the physicians have an increasing number of images to analyse manually [15]. In the practice, an ophthalmologist's diagnoses are not contested or validated by another specialist. Access to a system that generates a second opinion has an added value in serving as an extra pair of eyes without violating the autonomy, professionalism or credibility of the ophthalmology profession. Such a system contributes vastly to the decision-making process in all medical fields [15, 25], and it has greater odds of acceptance by specialists.

An example of an ophthalmologic disease is age-related macular degeneration (ARMD). This disease results in a deterioration of the central retinal function and is the leading cause of blindness in people over 65 years of age in Europe and the USA. Because of the localisation of the macula in the centre of the retina, advanced age-related macular degeneration often leads to irreversible loss of social skills, like reading ability. Two forms of ARMD are distinguished: the atrophic form and the neovascular, exudative or wet form. There are 3,000 such diseases in the field of ophthalmology.

This advanced cross-disciplinary DDDSN system has three services for practicing ophthalmologists:

- (1) Decision support image base
- (2) Continuous knowledge enhancement of the image base
- (3) On-demand learning module for specialists and residents to update their diagnosis and knowledge (see Table 1).

All fundus images that are acquired for clinical care at the institute are stored centrally in Topcon IMAGENet i-base [33]. It provides access to these stored images from workstations around the hospital. The system contains a database of both images and patient records. Table 1 is a summarised version of the application of the W*H specification frame for service systems modelling.

Our approach to service modelling supports the evaluation of a service in dependence on the answers to the W*H questions. The evaluation can be based on SWOT analysis that evaluates the benefit of some artifact to the environment. Figure 9 surveys the evaluation sheet for a health service based on the profile of the service, the opportunities provided and the threats. The health care is based on a task portfolio within the current situation. Therefore, we can select the most appropriate tactics. This evaluation supports communication between the stakeholders.

The model is now used for understanding what would be the benefit of the service, what kind of service can be expected, what changes must be made for an integration of the service and what is the added value. The typical situation is the nonexistence of a service model. Therefore, actors in health care act on partial information. This partial knowledge results in bad integration of a service and in waste of investment. Therefore, a service model will be an essential element of service deployment.

Table 1 Application of W*H for ARMD [2]

Service	Disease-diagnosis decision support	Knowledge enhancement and evolution	On-demand expert learning
(1) End (<i>wherefore</i>)	– Enhance diagnosis decision support for ARMD	– Evolve with new cases and treatments	– Enhance specialists and residents with knowledge.
(2) Sources (<i>whereof</i>)	– Repository of images and specific ARMD related descriptions	– Contributions of images from ophthalmologists	– Repository of specific ARMD descriptions
(3) Supporting means (<i>wherewith</i>)	– Computerized image comparison	– Computerized image submission	– Personalized learning wallets
(4) Surplus value (<i>worthiness</i>)	– Release specialist from tedious image by image matching	– Release specialist from depending on private image collections	– Learning whenever necessary at own time and speed
(5) Purpose * (<i>why</i>) * (<i>whereto</i>) * (<i>when</i>) * (<i>for-which-reason</i>)	– ARMD identification – Diagnosis – Early-stage-treatments – Second opinion	– New knowledge – Diagnosis – Early-treatments – Second opinion	– Learning – Diagnosis – Early-treatments – Enhance knowledge
(6) Activity * Input (<i>what-in</i>) * Output (<i>what-out</i>)	– Image of the patients eye – Matching image with diagnosis and treatment	– Image of new cases – Confirmation of image submission	– Request for a L wallet – Personalized learning module
(7) Party * Suppliers (<i>by-whom</i>) * Consumer (<i>to-whom</i>) * Producer (<i>whichever</i>)	– Ophthalmologist – Ophthalmologist – DDDS systems	– Ophthalmologist – Image repository – Image repository	– Image repository – Specialists/internists – On-demand L-envir-t.

<p>(8) Application domain</p> <ul style="list-style-type: none"> * Application area (<i>wherein</i>) * Application case (<i>wherefrom</i>) * Problem (<i>for-what</i>) * Organizational unit (<i>where</i>) * Triggering events (<i>whence</i>) * IT {data, control computation} * (<i>what</i>) * (<i>how</i>) 	<ul style="list-style-type: none"> - ARMD - During diagnosis - Treatment at early stages - Ophthalmologic unit - Successful match - Image comparison - Data 	<ul style="list-style-type: none"> - Maintenance - New knowledge - Disease evolution - Image repository - New submission - Knowledge enhancem. - Data 	<ul style="list-style-type: none"> - Expert Learning Evt. - On-demand learning - Personal-wallets - IT unit - Request for learning - Learning module - Data
<p>(9) Context</p> <ul style="list-style-type: none"> * System context (<i>whereat</i>) * Story context (<i>where-about</i>) * Coexistence context(<i>whither</i>) * Time context (<i>when</i>) 	<ul style="list-style-type: none"> - Location independent Ophthlrm.'s workspace - Impaired vision of patient - Integrateable i-base sys. - On-demand 	<ul style="list-style-type: none"> - Location independent workspace - New knowledge - Integrated DDDSN - On-demand 	<ul style="list-style-type: none"> - Location independent L-environment - Knowledge enhancem. - Integrated to DDDSN - On-demand

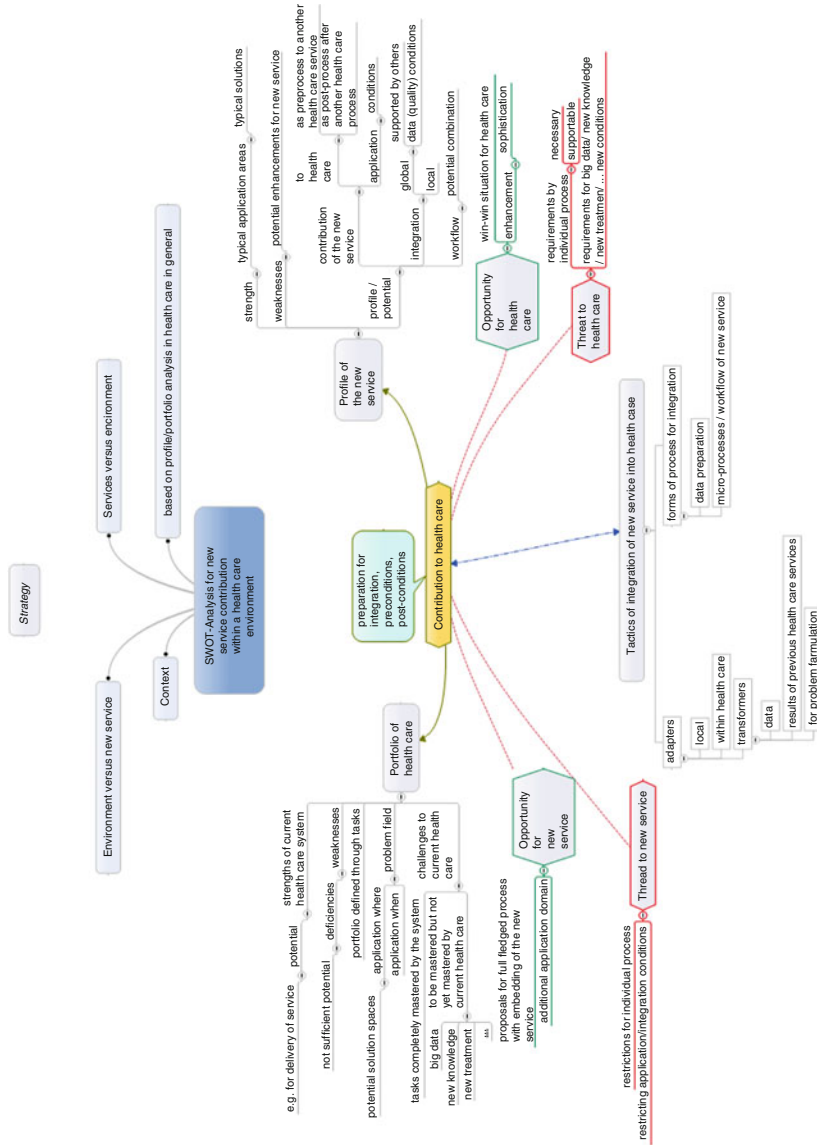


Fig. 9 SWOT analysis for a health service

References

1. Amarakoon, S., Dahanayake, A., Thalheim, B.: A framework for modelling medical diagnosis and decision support services. *Int. J. Digit. Inf. Wirel. Commun.* **2**(4), 7–26 (2012)
2. Amarakoon, S., Dahanayake, A., Thalheim, B.: Domain requirements modeling framework for cross-disciplinary healthcare service systems development. *Int. J. Healthcare Syst.* Palgrave Macmillan (The OR society). 2012, accepted for publication: ICCNDT, 152–166, Gulf University, Bahrain.
3. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. *IBM Syst. J.* **47**(3), 377–396 (2008)
4. Bergholtz, M., Andersson, B., Johannesson, P.: Abstraction, restriction, and co-creation: three perspectives on services. In: *ER 2010 Workshops of Conceptual Modeling of Services. Lecture Notes in Computer Science*, vol. 6413, pp. 107–116. Springer, Berlin (2010)
5. Dahanayake, A.: CAME: an environment for flexible information modeling. Ph.D. dissertation, Delft University of Technology (1997)
6. Dahanayake, A., Thalheim, B.: Conceptual model for IT service systems. *J. Univers. Comput. Sci.* **18**(17), 2452–2473 (2012)
7. Erl, T.: *SOA: Principles of Service Design*. Prentice-Hall, Englewood Cliffs (2007)
8. Fensel, D., Bussler, C.: The web service modeling framework WSMF. *Electron. Commer. Res. Appl.* **1**(2), 113–137 (2002)
9. Glushko, R.J.: Seven contexts for service system design. In: Maglio, P.P., et al. (eds.) *Handbook of Service Science, Service Science: Research and Innovations in the Service Economy*. Springer, New York (2010). doi:10.1007/978-1-4419-1628-0_11
10. Goldstein, S.M., Johnston, R., Duffy, J.-A., Rao, J.: The service concept: the missing link in service design research? *J. Oper. Manag.* **20**, 212–134 (2002)
11. Halloun, I.A.: *Modeling Theory in Science Education*. Springer, Berlin (2006)
12. Hirschheim, R., Welke, R.J., Schwarz, A.: Service oriented architecture: myths, realities, and a maturity model. *MIS Q. Exec.* **9**(1), 204–214 (2010)
13. Hurby, P.: *Model-Driven Design of Software Applications with Business Patterns*. Springer, Heidelberg (2006)
14. IBM Research: *Service Science: A New Academic Discipline?* IBM Press, New York (2004)
15. Ion, A.L., Udristoiu, S.: Automation of the medical diagnosis process using semantic image interpretation. In: *Proceedings of ADBIS 2010. Lecture Notes in Computer Science*, vol. 6295, pp. 234–246. Springer, Heidelberg (2010)
16. ISO/IEC: *Information technology - process assessment - part 2: Performing an assessment*. Publicly available, IS 15504-2:2003 (2003)
17. Maglio, P., Srinivasan, S., Kreulen, J., Spohrer, J.: Service systems, service scientists, SSME, and innovation. *Commun. ACM* **49**(7), 81–85 (2006)
18. McCarthy, W.E.: The REA accounting model: a generalized framework for accounting systems in a shared data environment. *Account. Rev.* **57**, 554–578 (1982)
19. OASIS: *Reference Model for Service Oriented Architecture 1.0*. <http://www.oasis-open.org/committees/download.php/19679/> (2006)
20. Papazoglou, M.P., van den Heuvel, W.-J.: Service-oriented design and development methodology. *Int. J. Web Eng. Technol.* **2**(4), 412–442 (2006)
21. Poels, G.: The resource-service-system model for service science. In: *ER2010 Workshops. Lecture Notes in Computer Science*, vol. 6413, pp. 117–126. Springer, Heidelberg (2010)
22. Preist, C.: A conceptual architecture for semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) *ISWS 2004. Lecture Notes in Computer Science*, vol. 3298, pp. 395–409. Springer, Heidelberg (2004)
23. Schewe, K.-D., Thalheim, B.: Development of collaboration frameworks for distributed web information systems. In: *Proceedings of IJCAI'07-EMC*, pp. 27–32 (2007)

24. Schewe, K.-D., Thalheim, B.: About semantics. In: 4th International Workshop, Semantics in Data and Knowledge bases 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6834, pp. 1–22. Springer, Heidelberg (2011)
25. Shortliffe, T.: Medical Thinking Meeting, London. Retrieved from <http://www.openclinical.org/dss.html> (June 2006)
26. Sol, H.G.: Shifting boundaries in systems engineering. In: Proceedings of the Second Pacific Asia Conference on Information Systems (PACIS), Singapore, 29 June–2 July, 1995, p. 26
27. Spohrer, J., Maglio, P.P., Bailey, J., Gruhl, D.: Steps towards a science of service systems. *IEEE Comput.* **40**, 71–77 (2007)
28. Stojanovic, Z., Dahanayake, A.: Service - Oriented Software Systems Engineering: Challenges and Practices. Idea Group Publishing, Hershey
29. Thalheim, B.: Towards a theory of conceptual modelling. *J. Univers. Comput. Sci.* **16**(20), 3102–3137 (2010)
30. Thalheim, B.: The theory of conceptual models, the theory of conceptual modelling and foundations of conceptual modelling. In: *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, Chap. 17, pp. 547–580. Springer, Berlin (2011)
31. Thalheim, B.: The science of conceptual modelling. In: *DEXA (1)*. Lecture Notes in Computer Science, vol. 6860, pp. 12–26. Springer, Berlin (2011)
32. Thalheim, B.: The conceptual model \equiv an adequate and dependable artifact enhanced by concepts. In: *Information Modelling and Knowledge Bases*, vol. 25, pp. 241–254. IOS Press, Amsterdam (2014)
33. Topcon: IMAGEnet i-base. <http://www.topcon-medical.eu/eu/producten/75-imagenet-i-base.html>. Accessed December 2014
34. Vargo, S.L., Lusch, R.F.: Evolving to a new dominant logic for marketing. *J. Mark.* **68**, 1–17 (2004)
35. Vargo, S.L., Maglio, P.P., Akaka, M.A.: On value and value co-creation: a service systems and service logic perspective. *Eur. Manag. J.* **26**, 145–152 (2008)
36. W3C Working Group: Web Service Modeling Language, Version 1.1. <http://www.w3.org/TR/sml/>, W3C Recommendation 12 May 2009
37. Zachman, J.A.: A framework for information systems architecture. *IBM Syst. J.* **26**(3), 276–292 (1987)

Monitoring of Client-Cloud Interaction

Harald Lampesberger and Mariam Rady

Abstract When a client consumes a cloud service, computational liabilities are transferred to the service provider in accordance to the cloud paradigm, and the client loses some control over software components. One way to raise assurance about correctness and dependability of a consumed service and its software components is monitoring. In particular, a monitor is a system that observes the behavior of another system, and observation points that expose the target system's state and state changes are required. Due to the cloud paradigm, popular techniques for monitoring such as code instrumentation are often not available to the client because of limited visibility, lack of control, and black-box software components. Based on a literature review, we identify potential observation points in today's cloud services. Furthermore, we investigate two cloud-specific monitoring applications based on our ongoing research. While service level agreement (SLA) monitoring ensures that agreed-upon conditions between clients and providers are met, language-based anomaly detection monitors the interaction between client and cloud for misuse attempts.

1 Introduction

Cloud computing [14, 163] industrializes service provisioning and delivery by offering infrastructure, platforms, and software components as rapidly deployable services. Especially technologies from web information systems have contributed to this success because of widespread availability on numerous platforms [104]. Clients can therefore consume cloud services or outsource applications to benefit from the scalability, elasticity, and computational power of private, public, or hybrid clouds. In this sense, a cloud service is considered to be a distributed, network-accessible software component that offers functionality to its clients [75, 139]. However, by utilizing a cloud service in a private or business-critical process, a

H. Lampesberger (✉) • M. Rady
Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University
Linz, Softwarepark 21, 4232 Hagenberg, Austria
e-mail: h.lampesberger@cdcc.faw.jku.at; m.rady@cdcc.faw.jku.at

client enters a dependency relationship with the service provider; a client actually needs assurance that a utilized cloud service is both dependable and correct.

Testing and formal verification are well-known methods to raise confidence in the correctness and dependability of a software component. Nonetheless, when consuming from cloud paradigms [14], such as infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS), a client faces cloud-specific challenges, in particular:

- The use of black-box software components in a cloud's execution environment, e.g., predefined libraries, procedures, and services, that have a specified input-output behavior but cannot be verified by the client and therefore lead to incomplete models [88];
- Partial or total loss of data governance [14];
- Operational nondeterminism [88] and randomness during runtime because a cloud's execution environment hides distributed computations from the client, necessary for elasticity and scalability properties of services, but eventually affects service execution;
- Cloud restrictions, e.g., technological and language restrictions, lead to insufficiencies [88] on the client- and service-side that require workarounds and eventually introduce untreated exceptional states;
- Client-side limitations to observe service deviations from normality during runtime, e.g., changes in the cloud's execution environment, hidden cases and assumptions, and cross-tenant information flow as a consequence of hidden sharing of computational resources.

The need for correct and dependable cloud services may seem obvious; however we highlight the economical dimension and why it is important. An incorrect cloud service could partially or completely fail because of an avoidable fault or an untreated exceptional state as listed above. The consequences are always business losses; these are usually costs associated with system repairs, acquisition and shipping of extra devices, external consulting, and additional working hours for employees, but there is also loss of customers and reputation. In addition, contract penalties for non-delivery of a service, legal fees, or costs for lawsuits could bankrupt a business.

Therefore, software deemed to be dependable and correct in a local environment eventually needs to be reevaluated when deployed as a cloud service or when parts are outsourced into the cloud. One method to reassure clients that a certain system behaves as specified is monitoring.

1.1 Monitoring

A *specification* captures the required functionality of some software, and *correctness* means that all input-output behaviors of an implementation satisfy the specification. Correctness is therefore a relative notion; software is correct with

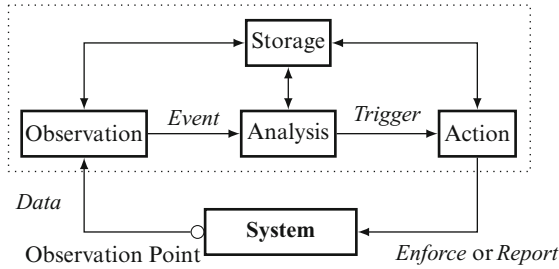


Fig. 1 A monitoring system has four main components: observation, analysis, action, and storage. An observation point represents technical means to recognize a system’s state and state changes. The observation component acquires data and generates *events*. The analysis component processes events and *triggers* the action component if necessary. The action component can then *enforce* actions or *report* events

respect to its specification. Avižienis et al. [16] define *dependability* in a broader sense as the ability to deliver a service that can justifiably be trusted. Dependability is qualified and quantified based on attributes such as availability, reliability, safety, confidentiality, integrity, and maintainability.

A *monitor* is then a system that observes the behavior of another system to reach some verdict about correctness: monitoring supports detecting, diagnosing, and recovering from failures and can furthermore expose additional state information for testing and debugging [49]. Our motivation is to understand if monitoring can also raise assurance of cloud service correctness and dependability.

A monitor performs four tasks that also characterize the major components as shown in Fig. 1 [49, 72]. The observation component of a monitor acquires data from an observation point and generates monitoring events for further analysis. Data acquisition is also referred to as *sensing* [166]. The analysis component implements a monitoring policy, i.e., the rules and conditions that need to be verified. If a violation is detected, the monitor triggers actions that can range from simple reporting to enforcement of certain behavior in the target system. Enforcement examples are simple termination, graceful degradation, recovery measures, and corrective interference. The storage component is accessed by other components for various tasks, e.g., to backup raw data or audit trails, as a knowledge base for analysis or for logging actions.

1.1.1 Monitoring Use Cases

Plattner and Nievergelt [143] argue that the fundamental motivation for monitoring is that a developer’s intuition about certain aspects of a system’s state space is more highly developed than the intuition about implemented code. Historically, monitoring of software components goes back to execution monitoring [143, 166],

and first applications have been debugging and performance enhancement but also checking correctness [165] and runtime verification [111].

Today, monitoring has a vital role in safety-critical distributed systems [72] and dependable systems [16], e.g., fault-tolerant systems [49, 159]. There are also numerous use cases to enforce a security policy in information systems, e.g., reference monitors for access control [160]. Also, intrusion detection categorizes monitoring applications that identify misuse attempts in program execution or computer networks [48, 50, 108]. With the advance of the World Wide Web, monitoring has also gained attention in web services (WS) [89], service level agreements (SLAs) [41], and recently cloud computing [2]. Aside from the diversity of monitoring applications, we identify the following use cases that are relevant for clouds:

- *Correct execution.* A monitor traces a cloud service's execution with respect to a policy, e.g., correctness conditions, and yields a verdict. Internal and external states are derived from observations and a monitor can therefore detect policy violations and eventually react to an unexpected state.
- *Dependability.* Implementations and hardware can be faulty. Unexpected, exceptional state transitions can also occur in cloud services, e.g., as a result of nondeterministic behavior due to dynamic scaling, pragmatic assumptions, or workarounds. Monitoring the service execution or interaction for violations and anomalies allows analysis and consequently fault control to raise dependability. Examples are failure prediction and root cause analysis [159] but also monitoring of security aspects.
- *Measurement.* Another monitoring use case is to determine the quality of service (QoS) by measuring (nonfunctional) requirements. Quantifying quality is relevant for cloud services, e.g., for automatic resource scaling or billing. Measurable SLAs are in fact necessary for automated contracting and controlling whether a promised service level is actually provided.

1.1.2 Monitoring of Client-Cloud Interaction

A monitor requires technical means, so-called observation points, to recognize the state and state change events of a system or service under observation. Observation points need to be chosen, so the states of interest, with respect to relevant properties in the specification, are unambiguously exposed. A monitor can therefore only observe a projection of the total state in accordance to the observation points in place. On the other hand, the available observation points restrict what properties can be effectively monitored. Cloud clients are directly affected because observation points in black-box software components on the provider-side are generally not accessible.

Furthermore, consuming cloud services is inherently a distributed computation task; a client needs to communicate with a service for interaction. Especially the PaaS and SaaS paradigms rely on communication standards found in today's

web applications, mashups, apps, mobile devices, and enterprise-grade services [104]. Client-cloud interaction therefore involves *languages* to encode information in a transportable format, standardized *protocols* to define message exchange over networks, and *architectures* that specify protocols as transport mechanisms and service interaction patterns, so clients can actually consume services. The communication between a client and a service is observable by its participants and can therefore offer observation points that are accessible to both client- and service-side.

The three central research questions with respect to client-cloud interaction are therefore as follows: “Why do we need to monitor?” “Where can we monitor?” “What do we monitor?”

1.2 Contributions and Structure

In this chapter we investigate observation points for client-cloud interaction and discuss two monitoring applications based on our ongoing research: SLA monitoring and language-based anomaly detection to monitor misuse attempts. More specifically, the main contributions are twofold:

1. We identify potential observation points in client-cloud interaction based on a literature review with respect to the state of the art of communication technology in SaaS and PaaS clouds.
2. We present our two ongoing research efforts as exemplary monitoring applications and discuss related work:
 - SLA monitoring is a measurement use case to check whether the quality of a system is compliant to the promised service level. The proposed approach defines an ontology for specifying SLAs for cloud services, based on which a monitoring system is developed. The monitoring system uses the ontology to identify the SLA conditions and detect if any violations of these conditions take place.
 - Language-based anomaly detection is an intrusion detection use case to identify anomalies, which are eventually caused by misuse. We consider message-based interaction between clients and clouds, where messages are expressed in the well-known Extensible Markup Language (XML) format.

The chapter is structured as follows. Necessary background information and definitions are recalled in Sect. 2 to outline the need for monitoring. We characterize quality attributes for services, service failures, origins of faults, security, and fault management, introduce necessary terms, and argue why monitoring can contribute to correctness and dependable clouds. Section 3 focuses on locations, where monitoring can take place, and potential observation points are identified. Our ongoing research efforts on SLA monitoring and language-based anomaly detection

are presented in Sect. 4 as two examples of what can be monitored in client-cloud interaction. Section 5 concludes the contribution.

2 Background

In this section we define and discuss different notions that are relevant to monitoring. The first notion we present is dependable computing and why it is important in the context of cloud computing. We also discuss correctness of services and service failures. We then introduce service quality and security.

2.1 Dependable Computing

In general, dependability is the ability to be trusted or relied on. In the context of computing, dependability is the ability to deliver a correct service that can be trusted and to avoid possible service failures. It includes different attributes such as *availability*, *reliability*, *safety*, *confidentiality*, *integrity*, and *maintainability* [16, 17]. Cloud computing introduces more challenges on *dependable computing*, since the client loses control over the management of the cloud. This increases the risk of uncontrolled outages that can affect the client's applications. The transparent sharing of cloud computing resources by different users can expose cloud applications to various risks. In addition, between the application and the cloud infrastructure, there can be different administrative domains, which reduces the visibility of the system and contributes to error propagation. As a result, problem detection can be very difficult. Hosting of private data off-premise may furthermore raise a lot of questions about privacy and confidentiality of the client's data. Last but not least, for competitive reasons, providers might not want to disclose information about their actual service level [91]. The notion of a correct service is discussed thoroughly in Sect. 2.2. In cloud computing service failure and unexpected exceptional states should be avoided; therefore we discuss service failures in Sect. 2.3, focusing on a system-centric view of service failures.

2.2 Correctness

The correct delivery of a software service is a well-discussed problem in the scientific community of dependable computing. We therefore follow the standard definitions and terminology of Avižienis et al. [16], Salfner et al. [159], and Goodloe and Pike [72]. Correctness of a software *system* is a relative notion with respect to some *specification*—a complete, precise, and verifiable description of requirements, behavior, and characteristics of a system [70]. A system delivers a *service* to a *client*

through its *interface* according to an agreed-upon *protocol*. The *total state* of a system captures computation, communication, stored information, interconnection, and physical condition [16]. The behavior of a system is then a sequence of states, a so-called *run*, such that the service is eventually delivered. The expected behavior is characterized by its functional specification. Every total state can be divided into an *external state*, perceivable at the system's interface, and an *internal state*; the sequence of external states of a system therefore represents the service.

Correctness. A system delivers *correct service* when the service implements the expected behavior, i.e., all possible sequences of external states conform to the specified input-output behavior. A transition to an incorrect service is a *service failure* and the duration of incorrect service delivery is a *service outage* [70].

2.3 Service Failures

The root cause of service failure is some *fault*. Figure 2 indicates how a fault in a system escalates into an *undetected error* that eventually becomes *detected* and possibly leads to *service failure*, where *symptoms* are side effects [15, 16, 49, 72, 159]:

Failure. A failure is the event when a service becomes incorrect such that the unexpected change of external state is observable by a user of the system, e.g., a client [16]. We also consider the transition into an untreated exceptional state [88] as a special case of failure. Things may go wrong inside the system, but as long as the external state conforms to the specification, there is no obvious service failure.

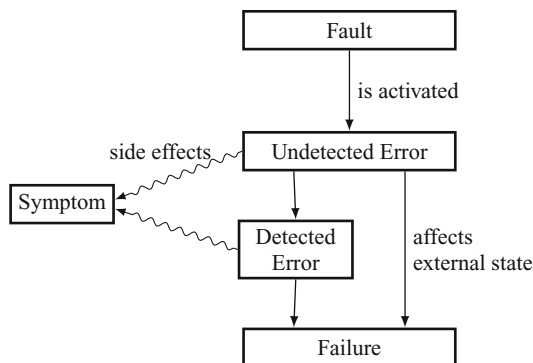


Fig. 2 According to the fault model of Salfner et al. [159], a fault stays dormant until it is activated and the internal service state becomes erroneous. The error eventually becomes uncovered by error detection routines. When the error affects the external state, it becomes a service failure. Errors can also cause side effects, so-called symptoms

Error. The cause of failure is when an external state becomes erroneous. An error is therefore the deviating part of the total state that may propagate to parts of the external state and lead to subsequent service failure [16]. Note that there are errors that only affect the internal state and cause no failure at all. We distinguish between *detected errors* and *undetected errors* [159]. An error is undetected as long as error detection routines do not report the erroneous state, e.g., by logging of assertions.

Fault. The cause of an error, i.e., root cause of a failure, is a fault [159]. Faults are *dormant* until they become *activated* and corrupt the system's state.

Symptom. Errors may not only lead to failure, they can also emit symptoms as a side effect, e.g., anomalous system characteristics [159]. For some errors, symptoms can already be observable before failure sets in.

Salfner et al. [159] also explicitly denote that there is an m -to- n mapping between faults, errors, failures, and symptoms; a single fault can lead to multiple errors and the same error could be caused by multiple faults.

2.3.1 Origins of Faults

Understanding faults is necessary to reach correctness and dependability in a software system, i.e., a web application or cloud service. Nonetheless, the root causes for software faults are manifold. Beside obvious mistakes in design, development, operation, or organization of a software system, e.g., software bugs such as missing deallocation of memory, infinite loops, or flawed interface usage, Jaakkola and Thalheim [88] identify five origins of untreated exceptional behavior and consequent service failure in implementations:

Incompleteness. Incomplete knowledge and coverage when specifying a system, macrodata modeling, integration of external libraries, and the inability to represent certain relations lead to a “modeling gap” [178] between specification and implementation. For example, the specification of a cloud service could deviate from an actual implementation because effectively used libraries in the cloud's execution environment are hidden.

Insufficiency. The insufficiency to represent current knowledge in the application domain is a result of implementation and conceptual language restrictions, restricted attention of developers, and locality of reasoning. Insufficiency leads to workarounds to partially patch or fix a situation, but they eventually introduce a stream of new exceptional situations.

Dynamic changes. Another root cause of service failure arises from negligence of evolution over time, e.g., caused by environmental changes, too restrictive models, unstable specifications, and temporary runtime errors.

Hidden cases. Neglected complexity and limited focus only on the “normal case” lead to various forms of assumptions, self-restrictions, and overlooked cases that eventually cause exceptional behavior.

Operational nondeterminism. An implementation is always executed in some environment. A (partially) hidden environment, automatic optimization, and changes in operational modes can lead to nondeterministic service behavior and randomness in a system’s run. For example, automatic cloud scaling could affect critical timing constraints in a cloud service.

Furthermore, cloud computing can exhibit characteristics of a distributed system and therefore inherit theoretical problems of distributed computing as a fault source. When synchronous or asynchronous interaction takes place, synchronization and resource allocation become challenging [113]. We recall *safety* and *liveness* properties that characterize temporal behavior of a distributed system [5, 107]: A safety property informally states that “*nothing bad will ever happen*,” while a liveness property states that “*(eventually) something good will happen*” in a run.

The faults in a distributed system are typically related to stopping and timing problems, e.g., race conditions or deadlocks, where synchronization of a critical section or resource fails. *Byzantine faults* are another temporal fault class, where a failing system does not fail in silence but starts to behave randomly, i.e., disobeys protocols, and communicates invalid messages to other systems [72].

2.3.2 Fault Management

Controls are necessary to achieve correct, dependable, and secure service delivery. Dependable computing literature [15, 16, 159] distinguishes four strategies for managing faults, and they are summarized in Fig. 3. *Fault avoidance* and *fault tolerance* are constructive; they aim for a resilient design and implementation of a system such that a specified service is delivered. Contrary, *fault removal* and *fault forecasting* focus on reaching confidence whether a system is able to deliver its specified service. Every strategy has a different goal:

Fault avoidance. Constructive methods, to prevent faults in the first place, are in the class of fault avoidance. Examples are coding guidelines, static code checking, design patterns, test-driven development, and constructive formal methods [15, 32].

Fault tolerance. Exception handling assumes that an exceptional situation can happen during a run and proposes routines to handle known exceptions and eventually recover [88]. Also, redundancy to free a system of single points of failure can achieve fault tolerance, i.e., a system has the ability to provide its

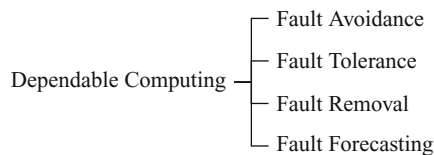


Fig. 3 The four fault management strategies toward dependable computing [15, 16, 159]

functionality even under the presence of faults [72]. Fault tolerance includes replication and fault-containment regions to ensure that an error does not propagate to other parts in a system.

Fault removal. The goal of fault removal is to minimize the number of faults in a system, e.g., by formal verification, model checking, and software testing [16].

Fault forecasting. Predicting the presence, occurrence, and consequences of faults is the goal of fault forecasting. Accurate forecasting of faults, errors, or failures enables proactive fault management, e.g., by online failure prediction [159].

2.4 Security

Security controls such as access control [160] or audits to uncover undetected errors ensure that a system respects its security requirements, but faults do not only affect correctness and dependability, they can also introduce insecurity. Systems and services have implicit or explicit security requirements that characterize allowed states and information flows in runs expressed as a security policy, e.g., access control rules. But faults can introduce undesired and unexpected transitions into an insecure state and therefore *violate* the security policy. We define the following security-relevant terms:

Threat. Any circumstance with the potential to affect systems, processes, or interactions in a way, such that security requirements are violated, is a threat. If the threat source is an agent, e.g., a person or group, we refer to it as *attacker*.

Exploit. A specially crafted input or interaction that leads a vulnerable system into an insecure state is called exploit.

Vulnerability. A fault becomes a vulnerability if (1) it is exploitable, (2) the threat has access to the vulnerable system, and (3) the threat has the capability of exploitation. An exploit is in fact a constructive proof of vulnerability existence in a system or service [162].

Attack. An attack, also called *intrusion* or *penetration*, is the actual process of an attacker exploiting a vulnerability. An attack becomes an *incident* when the attacker is successful and partially or fully reaches the attack goal. If an attack is novel, i.e., it exploits a system vulnerability that is not yet publicly known at the time of the incident, it is referred to as *zero-day (0-day) attack* [28].

Security of software is a deciding economical factor in future information systems [118]. Correctness of a system reduces the number of faults in a system and consequently mitigates the attack surface.

2.5 Service Quality

A complete specification of software covers both functional and nonfunctional requirements. Some requirements can receive higher attention because they are indispensable for correct function or *quality* of a service. Important quality

parameters are sometimes explicitly recorded in a contract called SLA. Today's SLAs are often written in natural language and are intended to specify the interaction and expectations, regulate resources, and define costs between the different parties involved in the agreement [145, 146]. Efforts are being made to specify the SLAs in a formal way. Important aspects of these SLAs are the set of quality parameters and the obligations of the different parties involved in the contract [92, 158].

2.5.1 Quality Attributes

In this section we define the different nonfunctional aspects. Some of them can actually be measured. We present here an informal compilation of a lot of different quality attributes that can contribute to the dependability, correctness, and security of a service:

Availability defines the readiness of the service to be used. A system is available if it is accessible when required for use [16, 145]. Availability contributes to the *dependability* of a system, if it is available whenever it is being accessed, and it gives an indication about the quality of the system *security*, if no vulnerabilities can be exploited to perform attacks on the system to cause unavailability.

Reliability is the continuous correctness of a system. It shows the ability of the system to operate without failure [16, 145]. It ensures the *dependability* of the system, if the system is being reliable. If *reliability* cannot be affected by security attacks, the *security* of the system is ensured.

Safety is the quality of averting or not causing injury, danger, or loss to the user [16] and contributes to the *dependability* of the system.

Confidentiality ensures the *dependability* of the system and is essential to the *security* of the system. It allows only authorized subjects to have access to the system [16, 145].

Integrity prevents improper alteration of system and system data [16, 145]. It ensures the *dependability* and *security* of the system.

Auditability is the ability to audit the system and to perform logging to verify its integrity, and it ensures both *dependability* and *security* of the system [195].

Privacy ensures that the user has control over sharing his/her personal information [145]. Privacy is a sub-property of *security*.

Authenticity is trusting the indicated identity or the integrity of message content [16, 145]. It is a sub-property of *security*.

Accountability ensures the existence and the integrity of a user performing a certain operation [15]. It is necessary for the *security* of the system.

Non-repudiability is to prove that users or systems cannot deny having sent or received a certain message [204]. It contributes to the *security* of the system.

Maintainability is the ability to maintain and repair the system easily and efficiently [16]. This ensures the *agility* of the system. An agile system can be changed and moved quickly and easily. It also contributes to the *dependability* of the system.

Modifiability is the ability to make changes to the system efficiently and with low cost [145]. This property contributes to the *agility* of the system.

Testability is the ability to test if the system is working correctly [145]. If the system is easy to test, then it is easier to debug and repair faults. This makes *testability* contribute to the *agility* of the system.

Usability is the ability to use the system easily and successfully [169].

Interoperability is the ability of different systems to work together. It is a *cloud-specific aspect* because cloud computing is evolving, and with the presence of different cloud providers, interoperability between clouds is of essence, so that complex systems can be developed from different available cloud services [138].

Portability is the ability to execute a program or run a system on different platforms and hardware systems [126].

Adaptability is the ability of the system to adapt its behavior to changes in its context.

Scalability refers to how the system can react and adapt to changes in system workload and varying needs [22].

Response time is a measure of system *performance* and refers to the amount of time the system requires to respond to a received request.

Throughput is also a measure of system *performance* and refers to the amount of operations the system can perform in a certain amount of time.

3 Observation Points in Client-Cloud Interaction

Communication is a central aspect of client-cloud interaction, and in cloud computing, the numerous standards for web and service technologies have started to converge. In this section, based on the review of monitoring literature and a survey of communication technologies [104], we enumerate potential observation points to monitor for service failures, errors, or symptoms for assessing the security and, in general, quality of a service in cloud delivery models.

When cloud services are consumed, a client transfers computational liabilities to a provider and loses some control over software components in the process. We distinguish two types of monitoring with respect to the client's capabilities to access or modify software components that need to be monitored: white-box and black-box monitoring. While white-box monitoring assumes that observation points can be accessed or added as necessary, e.g., by instrumenting code of a service, black-box monitoring infers a verdict about a service's dependability or correctness from observation points that are available to the client, e.g., network communication.

The categorization of observation points in terms of conceptual layers, as shown in Fig. 4, is motivated by the work of Spring [175, 176]. We specifically focus on the technological means that have been successful in related fields of software and are eventually available in today's clouds.

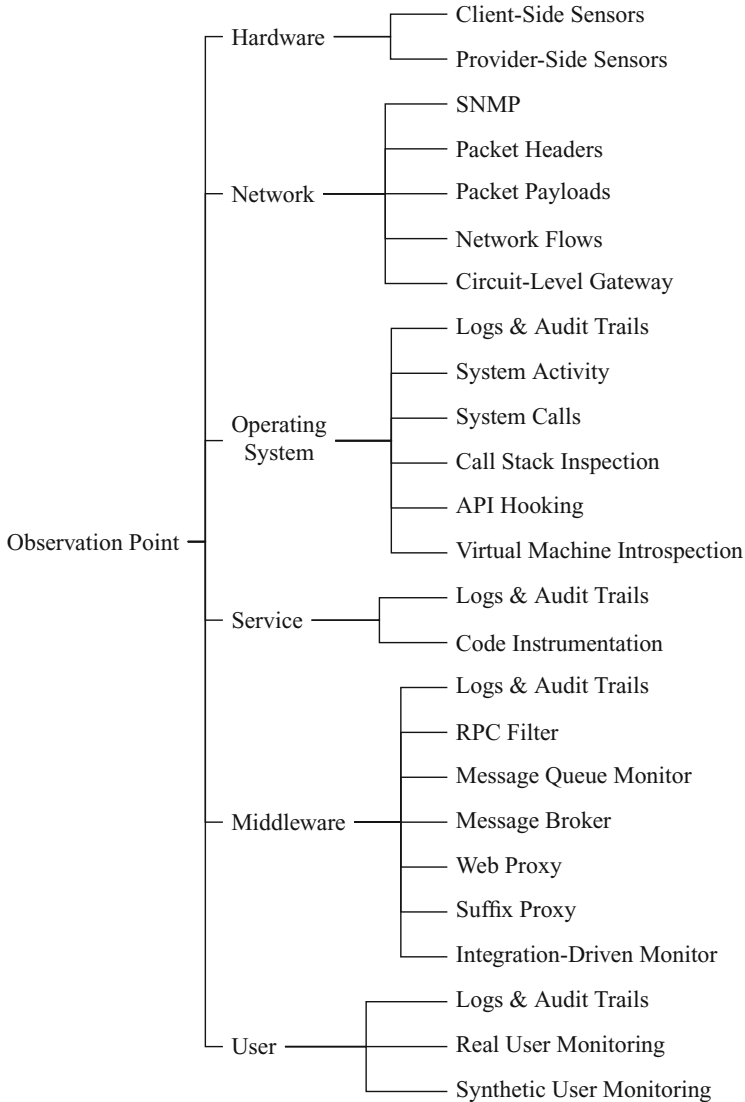


Fig. 4 An enumeration of potential observation points in client-cloud interaction monitoring

3.1 Hardware Layer

Computing hardware and devices typically offer various on-chip sensors to expose health information, e.g., system and processor temperatures, voltages, fan speeds, memory failure counters, hard disk health, performance counters, and clock speeds. But this information is typically not available in the cloud paradigm because physical hardware is hidden from the client and only accessible by the provider:

Provider-side sensors. There are cases where clients need hardware access in cloud computing, e.g., utilizing the graphics processing unit (GPU) for high parallel processing capabilities in Amazon Elastic Compute Cloud [8]. The service provider could expose hardware sensors to the client in such a scenario.

Client-side sensors. Client-side mobile devices can expose sophisticated sensory information through an agent for various monitoring applications, e.g., geographical location [73, 86] or device motion [11, 87].

3.2 Network Layer

Network communication offers various observation points when client- and service-side components are inaccessible for monitoring tasks. Communication protocols for information exchange in today's computer networks are *layered*; in particular, we consider the Transmission Control Protocol (TCP)/Internet Protocol (IP) stack [177] as the state of the art for network communication between clients and cloud services [104]. In packet-oriented Internet Protocol (IP) networks, only the header of an IP packet is required for routing decisions and delivery. When a system along the network communication path accesses the contents of a packet, beyond the IP header, this kind of access is called *Deep Packet Inspection* (DPI) [23], and we refer to these systems generally as middleboxes [34]:

SNMP. The Simple Network Management Protocol (SNMP) [79] is a client-server protocol for analyzing and managing network devices and it communicates over the User Datagram Protocol (UDP) for message transport. In an SNMP architecture there are agents running on network devices and a manager that retrieves or modifies variables in agents. Available variables are organized in a management information base (MIB) that describes a hierarchical namespace of object identifiers (OIDs). OIDs refer to device-specific variables, where variable values indicate configuration settings, device parameters, or network measurements [19, 182].

Packet headers. Besides the headers of IP packets, the Transmission Control Protocol (TCP) headers, UDP headers, and Internet Control Message Protocol (ICMP) messages in network traffic are observed. The packet payloads are left untouched.

Packet payload. A network monitor is payload based if also transport layer contents are inspected, i.e., payloads in TCP segments or UDP datagrams, using DPI technology. End-to-end encryption, e.g., by encryption protocols like Secure Sockets Layer (SSL) [64], Transport Layer Security (TLS) [51], or Datagram TLS (DTLS) [148], prevents payload inspection of packets in general. Payload-based observation can be further distinguished:

- *Packet level.* The monitor inspects the contents on a per packet basis. The fact that an application layer message can span over several packets is ignored [193, 194].

- *Reassembly.* The network monitor reconstructs complete or prefixes of higher-layer transmissions from packet traces collected from the network communication between a client and a service [105, 106]. Reassembly needs to consider properties of higher-level protocols; in particular, TCP segments can be out of order. Therefore, the monitor needs to keep track of all TCP connection states and eventually has to buffer packets [200].

Network flows. Network flows are summaries; they provide unidirectional or bidirectional meta information about network packets that share the same source and destination, IP address, ports, and IP protocol number [83, 199]. Any activity on the network layer creates flows, including UDP and ICMP. Collecting network flows raises less privacy concerns since no payload is observed. Furthermore, network flows can be collected from encrypted communication. Flows are exported by network devices as Cisco NetFlow [40] or IPFIX [83] records. An overview of flow data collection techniques in large-scale networks is given by Čeleda and Krmíček [35].

Circuit-level gateway. A circuit-level gateway is a control for transport protocols, e.g., TCP or UDP, and conceptually a middle man between communicating hosts [180]. A client establishes a session with a circuit-level gateway, and the gateway then forwards TCP segments or UDP datagrams from the client to the designated service, independent from higher-level protocols. The de facto standard for circuit-level gateways is Sockets Secure (SOCKS) [110].

3.3 *Operating System Layer*

Operating systems (OSes) are an essential building block in cloud computing architectures. The IaaS paradigm typically provides a virtualization environment for clients to operate individual operating systems [14]. Observation points are therefore of interest for both client- and provider-side monitoring:

Logs and audit trails. An operating system and its processes can produce various kinds of logs for failures, debugging information, notifications, and events in general. Log messages are then stored in files and databases or propagated over the network, e.g., by Syslog [71].

An audit trail in an operating system is a chronologically ordered sequence of security-relevant events. Audit trails are either exposed directly by the monitoring target or explicitly provided by the host operating system. Examples include access logs but also more sophisticated auditing tools, e.g., the Basic Security Module (BSM) [183] and the Linux audit framework [136].

System activity. A modern operating system typically collects runtime performance metrics such as system load, processor load, memory utilization, and network interface utilization. This information is exposed, eventually on a per process basis, and it is a valuable observation point for symptoms of failure. Examples are the `/sys` and `/proc` files in Linux systems.

System calls. A system call is a software interrupt triggered by a user-space process to request functionality from the operating system, e.g., to access files or start new processes. A modern operating system offers technical means to expose system calls during runtime for auditing or debugging [54, 66]. System calls can be acquired by modifications in the operating system kernel, *strace* in Linux [109] or *dtrace* in Sun Solaris and other Unix systems [137]; both tools additionally expose information about the target's call stack and program counter.

Call stack inspection. Every running process in a modern operating system has a call stack in memory that is composed of activation frames to remember nested function calls and to store local variables. Through modification of the operating system kernel, a monitor can also inspect the call stacks of processes during runtime [59].

API hooking. Related functions and procedures in software are often grouped in an application programming interface (API), e.g., the Windows API for operating system functions in Microsoft Windows systems [54]. A so-called *hook* is a function that intercepts and forwards an API function call for monitoring purposes. Therefore, API hooking enables analysis of function calls, call arguments, and eventually enforcement of behavior [54].

Virtual machine introspection. A monitor in an operating system is still a software component that could be vulnerable to attack or interfere with the monitoring target. To increase resiliency of monitors, the target operating system is virtualized and observed through a virtual machine monitor (VMM); this monitor can introspect the virtual hardware and memory, but it needs to reconstruct the state of the monitored operating system and its processes [52, 67, 130]. An example is runtime analysis of malicious software execution.

3.4 Service Layer

The software components that actually implement a service are of particular interest to be monitored. In case of cloud computing, these components are either provided by the client or operated by the service provider:

Logs and audit trails. Similar to operating systems, services can maintain individual logs for various purposes, e.g., service-specific events, performance counters, transaction logs for database systems, and access logs for auditing tasks to name a few. Logs are typically stored as a file or in a database.

Code instrumentation. To gather state information from a service, code instrumentation adds monitoring instructions to the service's code, so internal function calls, conditions, or data values become observable during runtime. Instrumentation can be done manually, by adding assertions or explicit pre- and post-conditions during implementation, or automatically.

Techniques for automated instrumentation are code rewriting [131], bytecode instrumentation [12, 29, 132], and insertion of advice statements for aspect-oriented programming [134, 135, 149]. Instrumentation techniques can be furthermore distinguished into static and dynamic, and instrumentation typically affects the performance of the software. An example for dynamic instrumentation of JavaScript code, popular in web-based applications, is given by Magazinius et al. [114]; script code is dynamically rewritten before its interpretation by the client. In addition to the two well-known applications of instrumentation, profiling and debugging, there are also enforcement of control-flow integrity [1] and information-flow security [157].

3.5 *Middleware Layer*

A middleware interconnects heterogeneous systems and services for integration purposes, and various types of middleware do exist in cloud computing [104]. Communication between clients and cloud services that passes through a middleware is often message based, and a number of observation points for monitoring are therefore available:

Logs and audit trails. Middleware solves various tasks, e.g., message routing and brokerage, coordinated actions, and service orchestration to name a few. Existing logging capabilities in middleware components, typically for debugging and auditing, are valuable observation points. Examples are performance and message brokerage logs [197], audit frameworks [196], and message flows [161].

RPC filter. Remote procedure call (RPC) architectures based on established web technologies are popular in cloud computing. Call argument data is serialized according to an agreed-upon data serialization method and typically has an individual media type in web-oriented RPC frameworks used in today's PaaS or SaaS clouds [104]. Filtering of arguments and return values when remote procedures are called is a potential observation point.

Message queue filter. Message queues enable asynchronous communication in message-oriented middleware [45]. A monitor can be placed as a filter in a message queue [4, 192] or as a message-processing component, so complete messages can be observed.

Message broker. When a middleware relies on dynamic routing [21] for message exchange, a monitor can participate as routing target or broker to receive and eventually forward message copies, e.g., by using WS-Addressing [185] in Simple Object Access Protocol (SOAP) [187] web services.

Web proxy. A web proxy is a networked software component that acts as a middle man. It forwards web requests and responses between a client and a service and eventually takes corrective actions by filtering or modifying content [98, 115]. Proxy architectures are popular for the Hypertext Transfer

Protocol (HTTP) used in today's web applications and services. We distinguish forward and reverse web proxy [179]: a client initiates communication through a forward proxy instead of directly communicating to the service. Contrary, a reverse proxy accepts messages on behalf of a specific service. A web proxy cannot observe contents of encrypted requests and responses in case of HTTP Secure (HTTPS) communication using SSL/TLS; encrypted interaction is transparently forwarded or blocked in such a case.

Suffix proxy. For web-based applications, a suffix proxy [115] is similar to a forward proxy: it acts as a middle man between client and service by forwarding client requests and gathering service responses. Contrary to traditional web proxies, a suffix proxy exploits the hierarchical naming scheme in the Domain Name System (DNS) to transparently and dynamically re-host existing services under a different DNS host name. For example, a suffix proxy for domain `suffix.org` could offer the services of `google.com` transparently over domain `google.com.suffix.org`.

A suffix proxy can observe and modify complete web requests and responses for monitoring purposes, and a use case is JavaScript code instrumentation for adding security controls [115]. Furthermore, a suffix proxy can access encrypted HTTPS communication when SSL/TLS client authentication is not performed. Nevertheless, all response messages passing through a suffix proxy have to be modified, i.e., adaptation of hyperlinks, so further requests are received.

Integration-driven monitor. Developers tend to reuse code, in particular, third-party libraries and other services to extend their application with additional features. When a middleware composes a particular service by integrating third-party libraries or services, the middleware can add observation points to monitor the consumed third-party libraries or services, i.e., an integration-driven monitor. Today's SaaS delivery models often rely on web technology, and web mashups are a popular approach to achieve composition and JavaScript code reuse [57, 174]. An example for an integration-driven monitoring in mashups is to instrument untrusted third-party JavaScript code [115].

3.6 *User Layer*

In web-based cloud services, monitoring of end users can deliver valuable insights for debugging or performance measurement. Besides existing logs and audit trails that expose user-centric metrics, we distinguish two complementary approaches for measuring end-user experience: synthetic and real user monitoring [44]:

Logs and audit trails. In PaaS and SaaS, the provided platform or software could already provide user-centric logging for auditing or service adaptation, e.g., user history, access, authentication, or geolocation logs.

Synthetic user monitoring. To monitor end-user experience, synthetic user monitoring actively simulates user requests using web browser emulation or recordings of web transactions. It emulates user behavior on a website and the different navigation paths, in order to measure performance and availability. Synthetic user monitoring distinguishes between internal tests that run locally within the organization or data center and external tests, where synthetic users are simulated over the Internet. While synthetic user monitoring can give a brief understanding about availability and performance of a service, it can hardly anticipate problems that could occur while the service is being used by many real users.

Real user monitoring. Contrary to synthetic user monitoring, real user monitoring is a passive approach to analyze every transaction of every user by observing actual interactions with the service. This allows to monitor whether a user is served in a timely manner and error-free and to detect the different problems in the whole business process. Real user monitoring captures, analyzes, and reports performance and availability of a service in real time as the visitors are interacting with it. Methods include sniffing, JavaScript injection, and installing an agent on the client-side [44].

4 Monitoring Applications

4.1 Service Level Agreement Monitoring

SLAs define assertions of a service provider that the offered service meets a certain guaranteed IT-level and business-process-level service parameters. In addition, the service provider should guarantee measures to be taken in the case of failed assertions [112]. Current SLAs in the market are written in natural language. It is the customer's responsibility to send claims of the downtime incidents, and the service provider checks if the claims are true. If the promised uptime percentage or rate is not met, the customer usually gets compensation in form of service credit [9]. Cloud computing imposes a challenge in this field, since the management of the SLAs is usually done by the service provider, who sometimes does not reveal the actual service levels of the service offering for competitive reasons. This is why the user should not only rely on the SLA received from the service provider, since it is not a reliable guarantee the actual service level is as promised. In order for the customer to be able to rely on the quality of the service, quality attributes need to be well described in SLAs. Their descriptions should be machine readable and allow efficient, accurate, and precise monitoring of the SLA in a client-centric manner, meaning that the client would also have control over how to manage and monitor these SLAs. This section discusses how to monitor SLAs from a client-centric perspective. Research done so far in this area has been unified on the need to formalize the specification of SLAs. Various attempts for the formalization

have already been introduced in literature, e.g., web SLA (WSLA), SLA language (SLAng), and SLA@SOI. All of them aim at formally specifying SLAs and making them machine readable to allow monitoring. Our approach is to try to define the SLA in a way that allows the user to monitor his/her own cloud system for SLA conformance or to outsource the SLA monitoring to a third party.

4.1.1 SLA Specification

In this section we investigate existing frameworks for SLA definitions. These frameworks are trying to find a general way to express all the quality aspects. The first model that we look at was developed by the SLA@SOI project. The SLA model in this project is concerned with modeling the physical structure of the document leaving out the intentional aspects of an agreement. The QoS term monitoring is in a later stage. The SLAs are then translated into operational monitoring specification. On this level are the intentional aspects of the contract tackled through special engines for this purpose [198].

Another way to model SLAs is using the SLA language (SLAng) [102]. The SLAng model defines two abstraction levels for compiling SLAs. It differentiates between vertical and horizontal SLAs. Vertical SLAs are concerned with governing the service level of the underlying infrastructure, while the horizontal SLAs are between parties providing services on the same level. SLAng is an XML for capturing SLAs. The SLA structure includes three main concepts: namely, an endpoint description of the contractors, contractual statements and QoS descriptions, and associated metrics.

WSLA is a framework for SLA establishment and monitoring of SLAs. The contract has three sections; parties, service description, and obligations. The WSLA language is XML based and SLA parameters are specified with their metrics [92].

The basic physical structure that the SLA should have includes the parties that are involved in the contract, the different guarantees or commitments that the service is offering, as well as some general information about the service itself.

Our efforts lie in trying to model the different quality aspects into depth, taking into account their individual meanings as well as putting in mind that the representation of the quality aspect should be monitored. Our approach to define SLAs [146] is using ontologies and the Web Ontology Language (OWL). OWL is based on description logic (DL). DL is a family of formal knowledge representation language and it is of particular importance in providing a logical formalism for ontologies and the Semantic Web. A DL models concepts, roles, individuals, and their relationships. In OWL we refer to these as classes, properties, and individuals, respectively. The different conditions of the SLA were modeled using DL axioms, these conditions will then be used to generate the contract, and since it is machine readable, it can be used by monitors to detect any violations. SLA monitoring is then discussed in Sect. 4.1.2. When defining an ontology, every class is a subclass of the class *Thing*.

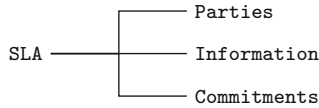


Fig. 5 Structure of the SLA document

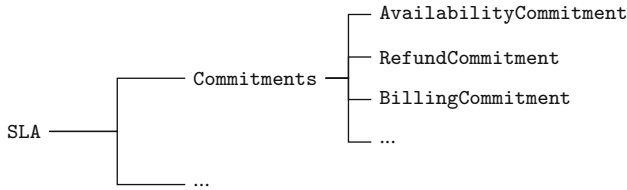


Fig. 6 SLA commitments

Figure 5 shows the structure of the SLA document. `Parties` define everyone who is involved in the SLA. `Information` describes the functionalities of the service; it lists all the possible operations to the service and the different failures as well as other relevant information about the service, such as the name or the service description. `Commitments` capture the nonfunctional aspects that are guaranteed by the SLA:

`Parties` are the different contributors that offer or receive a service. Each party has a name and a unified resource identifier (URI). Each SLA should have at least two parties involved. Parties can be a `Provider`, a `User`, or a `ThirdParty`.

$$\begin{aligned}
 \text{Parties} \sqsubseteq \exists \text{hasName.Datatype(string)} \\
 \quad \sqcap \exists \text{hasURI.Datatype(anyURI)}
 \end{aligned} \tag{1}$$

`Commitments` are the different conditions the service provider is offering to the customer. The first concept that is defined is `AvailabilityCommitment`. `RefundCommitments` and `BillingCommitments` are other defined concepts of the SLA. The ontology can then be extended to cover other commitments, by adding an attribute `X` as a concept `XCommitments` according to the quality specifications needed for the service (Fig. 6).

`AvailabilityCommitment` is the set of all commitments related to the quality attribute availability and is a subclass of `Commitments`.

$$\text{AvailabilityCommitment} \sqsubseteq \text{Commitments} \tag{2}$$

The subconcepts of `AvailabilityCommitment` are `CommitmentValidity`, `MaintenanceTime`, `ProbabilityDistribution`, and `MonitoringWindow`.

CommitmentValidity is one subconcept of AvailabilityCommitment, and it has the data type properties hasStart and hasEnd of data type datetime and has the object properties hasDuration and hasRepetition.

$$\begin{aligned}
 \text{CommitmentValidity} &\sqsubseteq \text{AvailabilityCommitment} \\
 &\sqcap ((\exists \text{hasStart.Datatype}(\text{datetime}) \sqcap \\
 &\quad \exists \text{hasEnd.Datatype}(\text{datetime})) \\
 &\sqcup (\exists \text{hasStart.Datatype}(\text{datetime}) \sqcap \\
 &\quad \exists \text{hasDuration.Duration})) \\
 &\sqcap (\exists \text{hasRepetition.Repetition})
 \end{aligned}
 \tag{3}$$

MaintenanceTime is another commitment that needs to be agreed on. It decides when maintenance will take place because the service might be unavailable for some time during maintenance. MaintenanceTime is a subconcept of AvailabilityCommitment. MaintenanceTime defines a start and end time for the maintenance. It has a property hasStart and hasEnd of data type datetime. In addition it has a relationship to the concept Repetition using the property hasRepetition that defines how maintenance is scheduled. For relative time, a start date and time and the duration of the maintenance are defined.

$$\begin{aligned}
 \text{MaintenanceTime} &\sqsubseteq \text{AvailabilityCommitment} \\
 \text{MaintenanceTime} &\sqsubseteq (\exists \text{hasStart.Datatype}(\text{datetime}) \sqcap \\
 &\quad \exists \text{hasEnd.Datatype}(\text{datetime}) \\
 &\quad \sqcap \exists \text{hasRepetition.Repetition}) \\
 &\sqcup (\exists \text{hasStart.Datatype}(\text{datetime}) \sqcap \\
 &\quad \exists \text{hasDuration.Duration} \\
 &\quad \sqcap \exists \text{hasRepetition.Repetition})
 \end{aligned}
 \tag{4}$$

Repetition is defined as:

$$\begin{aligned}
 \text{Repetition} &\sqsubseteq \text{TemporalInformation} \\
 &\sqcap \exists \text{hasRepetition.Datatype}(\text{integer}) \\
 \text{Repetition} &\equiv \{\text{Daily}\} \sqcup \{\text{Weekly}\} \sqcup \{\text{Monthly}\} \\
 &\quad \sqcup \{\text{Yearly}\}
 \end{aligned}
 \tag{5}$$

And Duration is defined as:

$$\begin{aligned}
 \text{Duration} &\sqsubseteq \text{TemporalInformation} \\
 &\quad \sqcap \exists \text{hasDuration.Datatype}(\text{double}) \\
 \text{Duration} &\equiv \{\text{Minutes}\} \sqcup \{\text{Hours}\} \sqcup \{\text{Days}\} \sqcup \\
 &\quad \{\text{Weeks}\} \sqcup \{\text{Months}\} \sqcup \{\text{Years}\}
 \end{aligned} \tag{6}$$

TemporalInformation is a helper concept used to extend the data type datetime with relative time.

ProbabilityDistribution is a subconcept of AvailabilityCommitment and defined as follows:

$$\begin{aligned}
 \text{ProbabilityDistribution} &\sqsubseteq \text{AvailabilityCommitment} \\
 \text{ProbabilityDistribution} &\sqsubseteq \exists \text{hasFormula.Datatype}(\text{string}) \\
 \text{ProbabilityDistribution} &\sqsubseteq \exists \text{hasParameter.Datatype}(\text{string})
 \end{aligned} \tag{7}$$

MonitoringWindow is the duration of time to which the availability commitment applies. MonitoringWindow is defined as:

$$\begin{aligned}
 \text{MonitoringWindow} &\sqsubseteq \text{AvailabilityCommitment} \\
 &\quad \sqcap \exists \text{hasDuration.Duration}
 \end{aligned} \tag{8}$$

RefundCommitment is the set of terms for getting a compensation in case the SLA is not met. The RefundCondition is the condition for the customer to receive a refund. If the service provides less service level than promised, the customer is entitled to receive a RefundPercentage. RefundCondition and RefundPercentage are defined to have a value of data type double.

$$\begin{aligned}
 \text{RefundCommitment} &\sqsubseteq \text{Commitments} \\
 \text{RefundCondition} &\sqsubseteq \text{RefundCommitment} \\
 &\quad \sqcap \exists \text{hasCondition.Datatype}(\text{string}) \\
 &\quad \sqcap = 1 \text{ hasRefundPercentage.RefundPercentage} \\
 \text{RefundPercentage} &\sqsubseteq \text{RefundCommitment} \\
 &\quad \sqcap \exists \text{hasRefund.Datatype}(\text{double})
 \end{aligned} \tag{9}$$

`BillingCommitment` represents the billing information. It has two sub-concepts: `Payment` and `Price`. `Payment` is a concept that defines where the payment is going to be made using a relation `hasURI` to the data type `anyURI`. In addition, it has a start date and time as well as a `Repetition` defining when the payment has to be made. The concept `Price` defines the price that has to be paid for the service, and the concept `Currency` defines the currency that is used to pay for the service.

$$\begin{aligned}
 & \text{BillingCommitment} \sqsubseteq \text{Commitments} \\
 & \text{Payment} \sqsubseteq \text{BillingCommitment} \\
 & \text{Payment} \sqsubseteq \exists \text{hasURI}.\text{Datatype}(\text{anyURI}) \\
 & \quad \sqcap \exists \text{hasStart}.\text{Datatype}(\text{datetime}) \\
 & \quad \sqcap \exists \text{hasRepetition}.\text{Repetition} \\
 & \text{Price} \sqsubseteq \text{BillingCommitment} \\
 & \quad \sqcap \exists \text{hasPrice}.\text{Datatype}(\text{double}) \\
 & \quad \sqcap \exists \text{hasCurrency}.\text{Currency} \\
 & \text{Currency} \sqsubseteq \text{BillingCommitment} \\
 & \text{Currency} \equiv \{\text{Dollar}\} \sqcup \{\text{Euro}\} \sqcup \{\dots\}
 \end{aligned} \tag{10}$$

`Information` is representing general information about the service and is a subtype of `SLA`. It includes different subconcepts (Fig. 7).

`Description`. This is a string defining what the service is used for.

$$\begin{aligned}
 & \text{Description} \sqsubseteq \text{Information} \\
 & \quad \sqcap \exists \text{hasDescription}.\text{Datatype}(\text{string})
 \end{aligned} \tag{11}$$

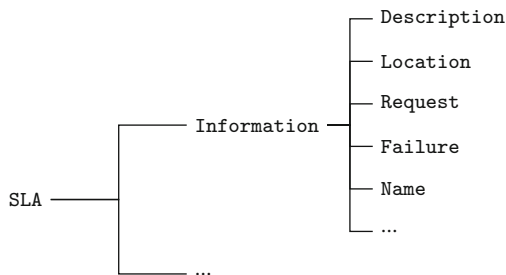


Fig. 7 SLA service information

Location. Any offered service should have a location. Not only the physical location to get information about jurisdiction but also the uniform resource locator (URL) or IP address under which the service is available and accessible.

$$\begin{aligned} \text{Location} \sqsubseteq & \exists \text{hasAddress.Datatype(string)} \\ & \sqcap \exists \text{hasIP.Datatype(string)} \\ & \sqcap \exists \text{hasURI.Datatype(anyURI)} \end{aligned} \quad (12)$$

Request. It describes the different requests that the client can make to different resources offered by the service. The availability of a resource can be modeled by a probability distribution function representing whether the resource was able to respond to the request made by the service user.

$$\begin{aligned} \text{Request} \sqsubseteq & \text{Information} \\ & \sqcap \exists \text{hasRequest.Datatype(string)} \\ & \sqcap \exists \text{hasFailure.Failure} \\ & \sqcap \exists \text{hasDistribution.ProbabilityDistribution} \\ & \sqcap \exists \text{hasMonitoringWindow.MonitoringWindow} \end{aligned} \quad (13)$$

Failure. This concept describes the different failures or errors the service can show.

$$\begin{aligned} \text{Failure} \sqsubseteq & \text{Information} \\ & \sqcap \exists \text{hasFailMsg.Datatype(string)} \end{aligned} \quad (14)$$

Name. This is a concept that has a string which is the name of the service. It is assumed here that the service name is a unique identifier, but if the name is not unique, another identifier can be used, such as the service URL or a specific ID. The Name relates a service to the different concepts in the ontology: MaintenanceTime, Request, Description, Parties, Payment, Refund, Location, Description, Price, MonitoringWindow, and CommitmentValidity.

All these concepts and relations form the knowledge base for the SLA conditions. After having the needed specification for the different conditions of the SLA, these should be monitored. In the next section, we will investigate different applications of SLA monitoring.

4.1.2 SLA Monitoring

The monitoring system needs to be aware of the structure of the system and its interaction model in order to monitor it. Each of the conditions stated in a SLA can have a measurable service level objective (SLO). This objective is the target that should be monitored. With cloud computing emerging, monitoring of SLA is becoming very important. Service providers have to pay penalties when SLAs are violated which can cause loss of money if the monitoring is not accurate. Currently there has been work done in the field of SLAs in cloud computing. However, none of these solutions have taken the lead on the market. This is why until now SLAs are stated in natural language, e.g., Amazon EC2 SLA [10]. There are different approaches to solve this problem: either to begin by defining SLAs and then set up monitors that are observing the parameters stated in the SLAs or by beginning with different resource metrics and trying to map them to SLA parameters. The proposed solutions use either their own monitors or suggest a service discovery for existing monitors.

A lot of the presented solutions for SLA monitoring are *agent-based monitoring* techniques. Agents are placed in the cloud system to gather measurements about the different quality of service aspects stated in the SLA [33, 39, 80, 125, 158]. The agents can be placed on the client-side, on the service provider-side, in the middle between the client and the service provider, or as probe nodes in the system [125]. Agents can also be placed in a combination of these locations.

Interest in SLA monitoring has been growing with the development of computing. First attempts introduce SLA monitoring for network and infrastructure. They use an agent-based monitoring technique [80] to monitor QoS parameters of SLAs from a customer's perspective. A flexible management agent (FMA) is placed in the network of the customer, and it observes network flows. They measure *availability* of IP connectivity and *response time* of the system.

Another contribution to SLA monitoring, presented in Choon et al. [39], uses an agent-based technique called aggregation and refinement based monitoring (ARM). This technique relies on detecting network flows that are violating the SLAs, especially in a large network. It is assumed that the network of an Internet service provider (ISP) has a large number of network devices; each of these devices can collect a large number of SLA flows and evaluate them through an agent that collects and sends measurement data. Because of the huge amount of flows, monitoring all of them is inefficient and might affect the system negatively. A data aggregation solution is provided to make monitoring more efficient and to achieve scalable monitoring. Data aggregation is the concept of using a controlled amount of data as an approximation for the actual service level. After the agent has collected QoS measures, it forwards an approximation of these measures to a Network Management System (NMS). The NMS forms a picture of the QoS that each SLA flow is showing. If the NMS needs more precise values, it will ask the agents to refine the results. This system can monitor the average throughput, packet loss, and packet delay of the network [39].

Ayad and Dippel [18] propose an agent-based monitoring system that checks for the availability of virtual machines and recovers automatically in case of failure. It is based on the idea of having multiple portable agents and a centralized monitor. The agents are configured in the different VMs they are monitoring. After configuration, the agent introduces itself to the centralized monitor. The task of the agent is to maintain information about the VM it is on and perform recovery actions such as restart, shutdown, or destroy. This is an architecture that can be applied for monitoring availability promised in SLAs.

It is particularly difficult to automate SLA monitoring of web services because they would need precise and unambiguous specifications and a monitor that makes measurements and reacts to certain events [158]. The first step was to create flexible and precise formalizations of what SLAs are. For the monitoring of SLA, attaching a proxy to listen to outgoing and incoming messages of the web service is difficult, as these are usually encrypted. As SOAP is one of the preferred standards for web service interaction, Sahai et al. [158] chose to modify and overwrite the SOAP toolkit to keep track of message exchange. There is a Web Service Monitoring Network (WSMN) agent that is loaded with the formal specifications of SLAs. The agent decides if the measurements should be done locally or externally. The measurements can be done on the customer- and provider-side, and the exchange of these measurements is done through the Measurement Exchange Protocol (MEP). Both sides need to agree on what kind of measurements to perform, the level of aggregation used, and how frequently the measurements should be exchanged. There are five different types of messages that can be exchanged by the protocol (init, request, agreement, start, report, close) [158]. If violations to the SLAs occur, violation records are stored in log files.

A framework called LoM2HiS is mapping lower-level resource metrics to higher-level SLAs. LoM2HiS maps states of the system, e.g., system up or system down, to a higher-level SLA term, e.g., availability. It uses the stand-alone Gmond module from the Ganglia open-source project as a monitoring agent on a host to retrieve the raw metrics from different resources. There is a runtime monitor that observes the service and compares the SLAs to the mapped metrics [55].

Another effort is the SLA@SOI framework [198], mentioned in Sect. 4.1.1. This framework, however, does not consider mapping the metrics to the specification, and the monitoring is done by special engines [198]. Rosenberg et al. [155] identify different QoS aspects from resource metrics. They present mapping techniques of the metrics to SLA parameters. However, they do not deal with monitoring of resource metrics.

Another proposed system that manages SLA violations is Detecting SLA Violation infrastructure (DeSVi) that allocates computing resources based on the user's request. The resources are monitored using the LoM2HiS framework, mentioned earlier [55, 56].

Romano et al. [154] propose a solution called Quality of Service MONitoring as a Service (QoS-MONaaS). The platform allows the user to define in a formal SLA the key performance indicators and sends alerts when these SLAs are violated. Dastjerdi et al. [46] rely on service discovery to find suitable monitoring services to

perform QoS monitoring. The monitoring service is described using Web Service Modeling Ontology, so the system is able to discover the suitable service.

A framework developed by International Business Machines (IBM), called WSLA, is used to specify and monitor SLAs for web services [92]. The WSLA framework comprises a definition of SLAs based on XML Schema and a runtime architecture providing different SLA monitoring services. This work presents a formal approach to SLA specification and runtime architecture for accessing the resource metrics and for monitoring and evaluation of SLAs. The SLA has a five-stage life cycle: negotiation and establishment, deployment, measurement and reporting, corrective action, and termination. The measurement and reporting stages deal with the computation of the SLA parameters by getting the metrics of the resources used for the application. There are two services responsible for this task: the Measurement Service and the Condition Evaluation Service. The Measurement Service measures the SLA parameters by either getting the resource metrics on the provider-side or by intervening with the client-side. The Condition Evaluation Service is responsible for comparing the retrieved metrics with the guarantees specified in the SLA. If a violation is detected, Corrective Measurement Actions have to be taken, and notifications are sent out to the client and the server. The component doing the monitoring was called SLA compliance monitor [92]; it was part of the IBM Web Service Toolkit. However, this toolkit is no longer available. It collects metrics from two points: it collects, first, directly from the managed sources inside the provider and, second, from outside the provider by issuing probing requests [125].

The first step for the monitoring of SLAs is SLA definitions. A formal language for defining SLAs should be used. SLAng is the SLA definition language used in Molina-Jimenez et al. [125]. For collecting the metrics agreed upon in the SLAs, a Metric Collector (MeCo) is installed where the interested party wants to have it. Four possible approaches to metric collection are presented. This is the decision on which *layer* the monitor will be placed. The first approach is the *service consumer instrumentation*, where the monitor will be placed on the consumer-side. The second approach is the *provider instrumentation*; the monitor is placed on the provider-side. There is also the possibility of placing MeCo at probe nodes; this approach is called *periodic polling with probe clients*. Finally, the monitor can be placed on the network and is called *network packet collection with request response reconstruction*. The last two approaches allow monitoring by third parties. The third party should perform measurement, evaluation and violation detection [125].

A patent presenting the idea of time-based monitoring of SLAs is describing how to monitor SLAs over defined time intervals [33]. The monitoring tests are defined by some data received; these are enforced at fixed time intervals. Users can specify when these tests can be run to check a particular level of service. The tests are distributed on different agents that are configured to communicate with the devices that are associated with the network. The SLA contains the type of network service, acceptable levels of performance, and a list of devices to which the SLA applies. They define the service level contract (SLC) to have a set of SLAs, and the SLAs are then configured to be run at a certain time. There is a service level manager (SLM)

server that gathers all the SLCs of all the customers to process and manage them. SLM serves as a central processing and reporting unit for SLC requests originating from clients. It offers an open interface to allow users to monitor and verify the level of service being provided. The standardized open interface can be provided through a document type definition (DTD) for XML, or a set of standardized template definitions can be used to define the different SLAs, what tests and when they will be enforced. The client can specify a set of metric tests that define the range of values that a certain service level can have. The SLM server is also responsible for managing the SLM agents. When the SLM server receives an SLC request from a client, it parses it and notifies the appropriate SLM agent to test the claims stated in the SLC. The SLM server regularly sweeps the SLM agent to get results about the tests and archives them. The SLM agent can use different functions for monitoring such as collecting data, performing data aggregation, monitoring resources, tracking non-responding devices, and maintaining data repositories.

Some of the solutions presented earlier rely on specifying the SLAs using XML which does not convey the semantics of the conditions of the SLA. Other solutions perform the monitoring on the provider-side only. Few efforts involve the client in the process. One of the challenging tasks in this field is bridging the gap between the SLA specifications and monitoring the SLOs and allowing the client to assess the performance and availability of the cloud in use. A step toward bridging this gap is shown in Rady [146]. In this work an abstract model was proposed to represent the interaction between the user and the cloud service that would allow modeling service availability in the SLA from a client-centric perspective, using the expressive power of description logics. The idea is to model the cloud service as a set of requests or operations sent to the cloud by the user, since this is how the user perceives the service. And the user receives for each of these operations a certain response. Each client can interact with a cloud service via a network. Each cloud service can be composed of a set of services, and the functionalities of the various services can be described as the set of all possible requests or operations that can be sent to the service. The availability and performance of these services can be monitored by observing the different requests and modeling their availability and performance using probability distribution functions. The SLA therefore reflects the architecture of the monitoring system, making sure that what the client or the third party is measuring is conveyed in the SLA.

In order to instantiate the SLA from the ontology, synthetic user monitoring is done to approximate the service level that is offered. On a client-controlled middleware, an SLA management component is placed to get the measurements from a monitor and save the service level information to the SLA knowledge base. Since the different conditions are stated in OWL, it allows the monitoring system to measure the service level and to add it to the contract efficiently. When the SLA is instantiated and the service is ready to be used by real users, real user monitoring is performed. It allows the monitoring system to automatically check whether there has been any violation of the conditions in the SLA and notifies the client immediately of the violation. The monitoring component is placed in a client-controlled middleware between the client and the cloud.

4.2 Language-Based Anomaly Detection

Another application of monitoring is to increase security of web and cloud services by monitoring the effective interaction between clients and services for attacks, so automated responses can be triggered. This section reviews intrusion detection and we focus specifically on anomaly-based detection techniques, applicable in web and cloud architectures, and discuss relevant problems in the domain.

Today's web and cloud architectures are built upon many heterogeneous technologies: TCP/IP and HTTP as transport mechanism; Hypertext Markup Language (HTML), JavaScript Object Notation (JSON), and XML to encode structured information; numerous media types; and interactivity through JavaScript [104]. We argue that XML-based attacks preferably exploit syntactic inconsistencies to cause insecure interpretation, i.e., unexpected tree structure or element content that leads to an insecure state. Therefore, we give an overview of our proposed *language-based anomaly detection* approach for XML-based interaction [103]. Detecting anomalous syntax can reduce the attack surface of XML processing systems on the client- and service-side.

4.2.1 Intrusion Detection

An *intrusion detection system* (IDS) monitors the information exchange or state of its target system to detect exploit activity caused by an attacker [162]. Intrusion detection techniques are typically deployed as software components or appliances, e.g., network firewalls, web security gateways, or web application firewalls. Notable surveys for intrusion detection are given by Debar et al. [48] and Lazarevic et al. [108]. When an IDS is capable of taking countermeasures, it is referred to as Intrusion Detection and Prevention System (IDPS or IPS).

In general, intrusion detection systems are categorized by the location of observation points into *host* and *network based*. We further characterize IDSs based on their detection strategy into *misuse-* and *anomaly-based* detection techniques.

Misuse-based detection relies on knowledge about attacks. An IDS compares observations with well-known behavioral patterns of exploit activity. A special case is *signature-based* detection, where an IDS matches for exact patterns of exploits in information exchange. Two notable signature-based IDSs for network packet inspection are Bro [140] and Snort [153]; both match the payloads of network packets against predefined rules and regular expressions.

Anomaly detection assumes that a deviation of normal behavior or input, a so-called anomaly, indicates a security violation. The history of anomaly detection goes back to the work of Denning [50], where an anomaly-based IDS has a reference model of the target system's *usual behavior* to distinguish observations into normal and abnormal events. There are two approaches for defining such a reference model: a formal specification of normal behavior, also referred to as *specification-based* anomaly detection, and by learning a reference model from observations or

measurements [37, 108]. The specification-based approach could be considered as an independent category, next to misuse and anomaly detection, if the specification also characterizes violating behavior. Learning a reference model in the second approach requires some form of training or learning period.

In theory, anomaly detection is more sensitive to false alarms than misuse detection, but anomaly detection has the advantage of recognizing yet unknown (*zero-day*) and *targeted attacks* that are specifically designed to evade signatures in misuse-based detection methods.

4.2.2 Survey of Anomaly-Based Intrusion Detection

To detect attacks in interaction, especially when black-box software components are in place, we look into monitoring techniques for observation points on the operating system, network, and middleware layer. In particular, we focus on anomaly-based intrusion detection intended for web applications and services, where higher-level protocol messages are exchanged [118].

Host-Based Anomaly Detection

Specifying and learning normal behavior for operating system audit trails or system calls are popular techniques in host-based anomaly detection. A specification-based anomaly detection technique for hosts is given by Ko et al. [94]. The proposed technique defines a program policy specification language to express an abstract security specification for a program, and a Unix tool then checks audit trails for conformance. This approach is further extended to distributed systems [95].

Wagner and Dean [190] present a specification-based anomaly detection technique that derives an IDS specification, represented as automata over system calls, directly from program code. Unfortunately, the runtime simulation of a derived nondeterministic pushdown automaton is intense, and efficiency is too low for many programs [59].

Another specification-based approach that relies on system calls is the Janus framework [66]. An OS kernel module intercepts system calls and checks them against a user-specified policy, and violating system calls are denied. Garfinkel [66] also highlights problems and pitfalls in monitoring system calls: incorrect replication of operating system state, indirect paths to resources, race conditions, and unforeseeable side effects when system calls are denied.

With respect to learning of a reference model for host-based anomaly detection, Forrest, Hofmeyr, and Somayaji [63, 82] introduce an IDS that instruments n -grams of system call sequences to learn a reference model of a monitored operating system process. A refinement of this approach delays or blocks system calls as corrective actions [170]. Both Sekar et al. [168] and Michael et al. [122] resort to automata learning techniques to infer profiles from *strace* audit trails. The *ViPath* model by Feng et al. [59] directly analyzes the call stack and return addresses of the

target program during execution to generate abstract execution paths between two program execution points. Another improvement in anomalous system call detection is Mutz et al. [129]; their proposed technique analyzes system call arguments and combines multiple detection models to increase IDS accuracy. The host-based IDS by Maggi et al. [117] clusters similar system call arguments and represents system call sequences by a stochastic model to evaluate runtime program behavior. Another stochastic model approach for system calls and arguments is given by Frossi et al. [65].

Anomaly Detection in Network Packet Headers and Payloads

For specification-based anomaly detection in networks, *eBayes TCP* [184] inspects packet headers and relies on a given Bayes network approximation of expected TCP network interaction. The model is then continuously refined during operation.

For network-based intrusion detection, where a reference model is learned, two notable systems are *PHAD* [120] and *ALAD* [119]. They observe TCP/IP network traffic and capture frequencies of certain properties in nonstationary probabilistic models. While *PHAD* models packet headers, *ALAD* additionally inspects the payload on a network packet-level and calculates normal frequencies of keywords.

Zanero and Savaresi [203] introduce an anomaly-based IDS that relies on unsupervised machine learning and a two-stage architecture. The first stage clusters similar network packets based on their contents, and the second stage detects anomalies in a rolling temporal window over clusters of packets.

The payload-based anomaly detector (*PAYL*), by Wang and Stolfo [193], estimates byte frequencies in network packet contents for a statistical reference model. A refinement of this approach, called *Anagram* [194], uses n -grams instead of single byte frequencies. *POSEIDON* [31] is a two-tier IDS, where packets are classified by a self-organizing map first, and a modified variant of *PAYL* analyzes mapped packets in a second stage for deviations. Perdisci et al. [141] further pursue this direction and introduce *McPAD*, an ensemble of one-class classifiers, to increase accuracy.

Another multiple classifier system that relies on hidden Markov models to capture sequences of content bytes in packets is *HMMPayl* [13]. To reduce the effort of generating training data, Görnitz et al. [74] redefine anomaly detection as an active learning task that queries an expert when detection confidence is insufficient. The proposed learning setting is intended for techniques such as *PAYL*, *Anagram*, or *McPAD*.

The denoted systems observe network packets and the applied techniques are independent from higher-level network protocols. Nevertheless, their experimental evaluations have included web interaction.

Anomaly Detection in Web Interaction

Kruegel and Vigna [97] introduce the first IDS for web applications that incorporates protocol syntax for anomaly detection. This IDS assumes that exploit activity in web applications manifests in HTTP header fields, especially in unified resource identifier (URI) query parameters. The proposed system observes and learns from HTTP request headers, and the detection technique uses a linear combination of six different anomaly detection measures, like attribute character distributions, structural information, or attribute lengths. The system of Kruegel and Vigna [97] is a foundation for follow-up research: Robertson et al. [151] generalize anomalies by inferring attack types using heuristics; Maggi et al. [116] address the impact of the changing nature of web applications, a phenomenon called *concept drift*; and Robertson et al. [152] deal with scarce training data by reconciling a knowledge base, while the reference model continuously improves.

Ingham et al. [85] also instrument HTTP headers for anomaly detection. Their algorithm learns a finite automaton representation from tokenized web request headers. Düssel et al. [53] detect deviating web request headers using support vector machines (SVM), where feature extraction and SVM kernel incorporate application layer syntax. *Spectrogram* [173] reassembles bidirectional TCP network streams between client and service, and multiple Markov chains evaluate URIs in HTTP GET requests or message bodies in HTTP POST requests. Hidden Markov model web (*HMM-Web*) [42] instruments multiple hidden Markov models to analyze URI paths observed in HTTP GET requests. Lampesberger et al. [105, 106] propose an anomaly detection system for URI paths in web requests based on a variable-order Markov model that continuously learns without a separate training phase.

To detect widespread simultaneous zero-day attacks, Boggs et al. [30] propose a distributed architecture of content anomaly detectors situated at different websites. The individual detectors implement the *Anagram* technique [194] and exchange abnormal content for mutual evaluation. This approach can reduce false-alarm rates for rare and high entropy web requests.

The IDS proposed by Kirchner [93] operates as an HTTP reverse proxy and it inspects both web requests and responses. A clustering algorithm adjusts the reference model to the actual usage patterns during operation. *TokDoc* [98] is another HTTP reverse proxy approach for protecting web servers. An ensemble of anomaly detection methods detects and automatically repairs malicious web requests.

In service-oriented paradigms like web services [4] and RESTful services [62], HTTP degrades merely to a transport mechanism for messages and attacks manifest in message content [118]. Criscione et al. [43] introduce *Masibty*, an IDS for web applications that acts both as reverse proxy for HTTP and a monitor for Structured Query Language (SQL) database calls. The proxy instruments several anomaly detection modules that analyze not only HTTP headers but also the message content sent in web requests and responses. If a web request causes a database query, the *Masibty* system checks whether the SQL query and response are expected. Therefore, the client-side is protected to a certain extent from malicious responses sent by the server.

Rieck [150] proposes a machine learning framework for detecting unknown attacks on the application layer, i.e., communicated messages. The framework covers a generic technique for embedding any kind of network stream in a vector space, kernel functions for learning in high-dimensional vector spaces, and learning methods for geometric anomaly detection. The framework by Krüger et al. [99] also has applications in anomaly detection; the presented method automatically extracts semantically relevant sections of network traffic using vector space methods. The extracted sections are potential features for anomaly detection algorithms.

For defenses against distributed denial-of-service (DDoS) attacks on web applications, Xie and Yu [201, 202] propose anomaly detection in user browsing behavior. User references and resource descriptions are extracted from HTTP headers, and an extended hidden semi-Markov model captures the browsing behavior of regular users as reference model.

4.2.3 Intrusion Detection Problems

Intrusion detection systems operate in adversarial environments, where attackers try to evade detection or even attack monitoring components directly [171]. Ptacek and Newsham [144] demonstrate how DPI-based network monitors can be evaded. Besides extensive packet fragmentation, when the number of network routing hops to the target is known, an attacker could insert bogus packets when interacting with the targeted system. The bogus packets are dropped by Internet routers before reaching their destination, but an unaware IDS observes a misleading sequence of packets and eventually misses the attack. Handley et al. [78] present a countermeasure, but this fundamental problem still prevails in many DPI-based systems today [133].

Polymorphic exploits have a long-standing history in evading signature-based detection techniques [172]. Signatures for network-based intrusion detection are typically syntax based, and polymorphism expresses exploits syntactically different, while attack semantics are preserved.

Wagner and Soto [191] coin the term *mimicry attack* for anomaly- and misuse-based detection techniques. With respect to available observation points, a monitoring system has a projected view on the actual system state or exchanged messages. A knowledgeable attacker could adapt an exploit, so it perishes in the projection. For example, suppose an anomaly detector maintains a stochastic reference model of byte frequencies in normal network packet payloads. An attacker can evade detection by padding an exploit with random bytes, so frequencies are indistinguishable by the anomaly detector.

Anomaly detectors are also vulnerable to *poisoning* during training [156]. If an attacker is able to influence the learning process or hide attacks in training data, future attacks could be assumed as normal in the reference model. This raises the problem of attack-free training data and requires robust anomaly detectors. Chan-Tin et al. [36] formalize the *frog-boiling attack* as a special case of poisoning against a monitor that continuously learns. A knowledgeable attacker sends messages that

are borderline normal, but they falsify the reference model over time. At some point the reference model will assume actual attacks as normal.

Poisoning indicates that a learning algorithm can be a target of an attack. Barreno et al. [20] discuss different types of attacks against machine learning algorithms and propose countermeasures to limit the capabilities of an attacker. This problem has given rise to resilient learning schemes and research in adversarial machine learning [84], where an attacker is assumed to interfere with the learning process.

Formal Language-Theoretic Security Problems

In a recent study by Sassaman et al. [162], the authors discuss software vulnerabilities using formal language theory. A protocol basically specifies the syntax and semantics of a formal language for encoding information, e.g., invocation arguments, file formats, or network messages. When two components R and S interact, the sender S encodes information with respect to the protocol as transportable message. The receiver R interprets (*parses*) the message according to the protocol and updates its system state in the process. But a real-world protocol and its implementation can suffer from faults, and faults can *unexpectedly* increase the expressiveness of the protocol [162]; a sender S might be able to craft an exploit such that R moves into an unexpected or insecure state upon parsing.

To resolve vulnerabilities, an unambiguous protocol specification and a fault-free implementation are required, so the receiver can reject invalid messages. In terms of formal languages, rejecting messages is a membership decision problem, and based on the expressiveness of the protocol, it is eventually intractable or undecidable.

This fundamental problem also affects many IDSs because they are typically engineered for a tractable detection method that implicitly assumes a formal language class of the observed data [162]. For example, *regular word languages* (REG) are popular in signature-based intrusion detection because deterministic matching has constant space and linear time complexity. The goal of intrusion detection is nevertheless to detect exploit activity that is eventually beyond the language class of the method or across layers of interleaved protocols. If detection methods in an IDS are not as expressive as the formal language class of observed data, the consequences are false alarms and false-negative detections. For example, suppose that the IDS operates in REG and the protocol language class is more powerful than REG; there could be infinitely many exploits that are still considered normal with respect to REG. Respecting the language-theoretic problems is therefore indispensable for anomaly- and misuse-based detection methods [162].

Hadžiosmanović et al. [77] experience the problem of mismatched language classes in their experimental evaluation, where several anomaly detectors based on n -grams cannot deliver acceptable accuracy for real-life binary network protocols. The class of n -grams is a strict subset of regular distributions [47] and expressiveness is limited; n -grams over bytes cannot correctly approximate the complex binary protocols, where individual bits have symbolic character. An anomaly detector has

to respect the symbolic and structural characteristics of a protocol language to achieve acceptable results.

4.2.4 Language-Based Anomaly Detection in XML-Based Interaction

Today's web and cloud services rely on message exchange for interaction over various transport mechanisms, and XML has become a popular format to serialize information in a transportable format, e.g., asynchronous JavaScript and XML (AJAX) [69] in web applications or SOAP in web services [104]. We therefore propose language-based anomaly detection to discover attacks in interaction and identify the following characteristics:

Respecting the language class. Mismatching formal language classes in detection algorithm and observation point lead to bad detection performance. Low false-alarm rates and high detection rates at the same time can only be achieved when language classes are respected. We focus specifically on XML, a semi-structured language and essential for many protocols in today's web and cloud services [104]. Identifying anomalies in XML documents can reduce the attack surface of systems that rely on XML processing.

Message-level monitoring. As argued in Sect. 4.2.3, inspection of network packets or reassembly of application data from packet traces can be evaded by a skilled attacker. Furthermore, network interactions are more and more encrypted by default, which renders DPI-based observation unpractical [104]. This issue can only be dealt with if an IDS observes the message level of interaction on the middleware layer, e.g., as a web or suffix proxy, message broker, or message queue filter. The anomaly detector for our proposed system specifically analyzes XML documents.

Grammatical inference. Given the language-theoretic understanding of protocols, we see learning of a reference model for anomaly detection as a special case of grammatical inference [47]: to learn a formal language *representation* from its *presentation*. A grammar or automaton is a typical representation, where a set of examples, counterexamples, and an oracle are presentations. Grammatical inference assumes that there is a *hidden target* representation to be discovered, and language class and type of presentation influence successfulness of learning [47, pp. 141–172]. A learning algorithm is said to converge if the hidden representation is uncovered. Convergence in an anomaly detector implies that false alarms and false-negative detections will be minimal in the long run.

Figure 8 outlines our problem definition: Given a set of example of XML documents, a grammatical inference learner returns an automaton as reference model. The automaton *validates* the *syntactic structure* and *data types* of future documents, and an accepted document is considered normal with respect to the training data.

The underlying logical model of XML is a tree and processing it as *Document Object Model* (DOM) [186] requires a full representation in memory. This becomes

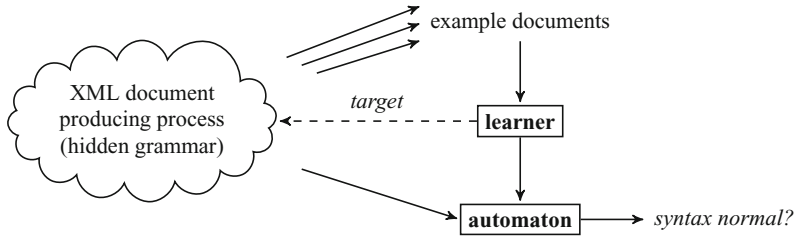


Fig. 8 From example documents the learner infers an automaton that validates future documents

hard with increasing size of documents and impossible for continuous XML streams. We therefore require automaton and learner to process streams, where memory and time are limited. Our streaming interface to documents is the *Simple API for XML* (SAX) [181], and *visibly pushdown automata* (VPA) [6, 101] are an appropriate language representation of XML capable of stream validation.

XML-Based Attacks

XML can express complex structures, but parsing is also complex and faults can happen. DOM parsers read the whole document into a logical tree form—they are vulnerable to denial-of-service (DoS) attacks that target time and memory, e.g., by *overlong element names* or *oversized payload*. Also, nesting many tags can cause DoS and this is referred to as *coercive parsing attack* [58].

The preamble of a document can define or point to a DTD. Several DoS attacks against DTD-respecting parsers are based on recursive *entity expansion*. A parser can also violate confidentiality of the system if *external entity references* allow local file import [58].

When the attacker has control over parts of a document, a potential attack is *XML injection*. The fictional document in Fig. 9a that describes a monetary transaction is a running example; a value is predefined, and the user has control over the credit card number section. Some DOM parsers are vulnerable when user-submitted tags override existing elements, like in Fig. 9c, where the attacker manipulates the transaction value [58]. Bad state handling in a SAX parser can also lead to XML injection vulnerabilities. Another popular injection attack in the web is *cross-site scripting*, where an attacker embeds an illicit JavaScript or Iframe in the document. Also classic attacks like *SQL* (Fig. 9b), *command*, or *XPATH injection* can be embedded in XML element content to target other components in a system.

Schemas and Validation

The discussed attacks change the expected document syntax—unexpected structure or data types could exploit XML processing or other components in a vulnerable

a	b
<pre> <transaction> <total>1000.00<total> <cc> 1234 </cc> </transaction> </pre>	<pre> <transaction> <total>1000.00<total> <cc> 1234' or '1'='1 </cc> </transaction> </pre>
c	
<pre> <transaction> <total>1000.00<total> <cc> 1234</cc><total>1.00</total><cc>1234 </cc> </transaction> </pre>	

Fig. 9 (a) Expected format of an example; an attacker controls credit card number [58]. (b) SQL injection attack that violates the expected credit card data type. (c) XML injection attack specifically for DOM parsers [58]

system. For specifying and verifying structural rules of documents, XML offers *schemas* and *validation*. A schema is a grammar that characterizes the logical tree structure of acceptable XML documents, and validation is to check if an XML document conforms to a given schema.

There are several schema languages for expressing schemas, e.g., DTD [186] and its generalization *Extended DTD* (EDTD) [121], the industrial standard *XML Schema* (XSD) [188], and *Relax NG* [127], to name a few, and they have different powers of expressiveness [24, 121, 128]. Stream validation is then to accept or reject a document in a single pass, i.e., a membership decision in terms of formal languages. Validation can reject documents with unexpected syntax and therefore greatly reduce the number of vulnerabilities in XML processing [58, 90].

Unfortunately, validation is not common in the web. In a study by Grijzenhout and Marx [76], only 8.9% of documents in the web refer to a schema and are valid. Schemas are often too general or outdated, and paradigms like AJAX do not enforce validation, so developers tend to ad hoc design. Learning a language representation as reference model from effectively communicated XML is therefore a promising approach.

Related Work

There is a large body of XML research in stream validation and schema inference. Our presented system is to our knowledge the first that aims for both stream processing and VPA inference including element content data types. With respect to XML stream validation, the first discussion is given by Segoufin and Vianu [167]. To capture the entire class of regular tree languages in stream validation, Kumar

et al. [101] introduce VPA for XML (XVPA). Schewe et al. [164] extend VPA for approximate XML validation, and Picalausa et al. [142] present an XML Schema framework using VPA.

We direct the reader to the book of de la Higuera [47] for a survey of grammatical inference. We apply the concept of function distinguishable languages [61] for our inference algorithm [103]. Kumar et al. [100] argue that VPA can be learned in a query learning setting that differs from our setting in the problem definition.

Inference of schemas goes back to Ahonen's work on Standard Generalized Markup Language (SGML) [3]. Inference of DTDs from XML is well researched [26, 27, 60, 68], but the problem becomes harder for the more powerful language class of XSD. Mlýnková [123] gives a survey of XSD inference: the general approach is to first infer an extended context-free grammar from examples and then merge nonterminals [124]. With respect to data types, Hegewald et al. [81] and Chidlovskii [38] also consider approximations of XML content via data types. Bex et al. [25] rely on tree automata for learning l -local Single Occurrence XSDs in a probabilistic setting; our approach is similar but focuses on stream processing. Kosala et al. [96] and Raeymaekers et al. [147] describe information retrieval algorithms to learn wrappers for HTML as tree automata.

4.2.5 A Learning Algorithm for Language-Based Anomaly Detection

We now sketch the grammatical inference algorithm for language-based anomaly detection. For detailed information we direct the reader to Lampesberger [103]. Inference of an automaton as a reference model for detecting anomalous syntax in XML is done in three steps:

1. *Document abstraction.* We define rules that transform an XML document into a stream of SAX events. XML content, in particular strings between tags, is reduced to a finite set of data types.
2. *Visibly pushdown prefix acceptor (VPPA).* In the second step, a special kind of pushdown automaton, called VPPA, is constructed from a set of example documents, i.e., the training data.
3. *State merging.* To generalize knowledge from training data, states in the VPPA are merged. The resulting automaton is finally converted into an XVPA for stream validation of future XML documents.

Document Abstraction

The SAX processing model for XML emits events when an XML document is parsed. We consider the SAX events *startElement* for open tags, *endElement* for close tags, and the *characters* event for element content. Attributes are considered as elements to preserve the logical tree structure, and namespaces are directly embedded in element names. XML content requires special treatment because

$$\begin{array}{c}
 \langle a \rangle \langle a \rangle 10.0 \langle /a \rangle \langle b \rangle \text{some text content} \langle /b \rangle \langle b \rangle \langle /b \rangle \langle /a \rangle \\
 \Downarrow \\
 aa\{\text{decimal}\}\bar{a}b\{\text{string}\}\bar{b}b\bar{b}\bar{a}
 \end{array}$$

Fig. 10 An XML document is abstracted as a stream of SAX events, where the lexical data type system reduces actual text contents to a finite number of data types

the language class of some string between two tags is unknown and learnability properties are affected.

We therefore introduce a *lexical data type system* in Lampesberger [103] for abstracting content by data types. XSD provides a rich set of data types [189], where every data type has a semantic value space and a lexical space, i.e., allowed strings over the Unicode character set. A learner only observes the lexical space; every string between two tags in a document is therefore reduced to a single SAX event (*characters*) that contains the set of matching data types. Based on this abstraction, a document becomes a stream of events, and an example is shown in Fig. 10.

Visibly Pushdown Prefix Acceptor

XML is a visibly pushdown language because tags have to be well matched [6, 7], and XVPA are an appropriate language representation [101]. XVPA are special pushdown automata that have three disjoint alphabets for SAX events in our definition: *startElement* events, *endElement* events, and data types for *character* events, so they are compatible with our abstraction of documents. Furthermore, the stack alphabet is the set of automaton states. Every schema has an equivalent XVPA, and stream validation is then the acceptance of a document's event stream by the corresponding XVPA [101].

For grammatical inference, the intuition is that every prefix of every document in the training set leads to a unique state in the VPPA. All training documents are passed once to construct a VPPA, a pushdown automaton that accepts exactly the documents in the training set, and an example is shown in Fig. 11. State merging then generalizes the automaton.

State Merging

To generalize the observed language of training documents toward the hidden schema of the XML document producing process, states in the VPPA are merged. We define a so-called distinguishing function that indicates whether two states are equivalent and can be merged. The resulting automaton has fewer states, generalizes the language, and is converted into a valid XVPA as a reference model. This XVPA is capable of stream validating future documents to detect anomalous structures or data types with respect to the language exhibited by the training data. The

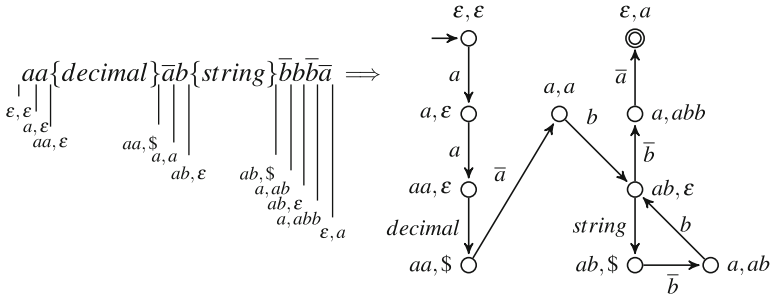


Fig. 11 Every prefix of an abstracted document defines an individual automaton state. The VPPA is then the automaton constructed from all prefixes of example documents

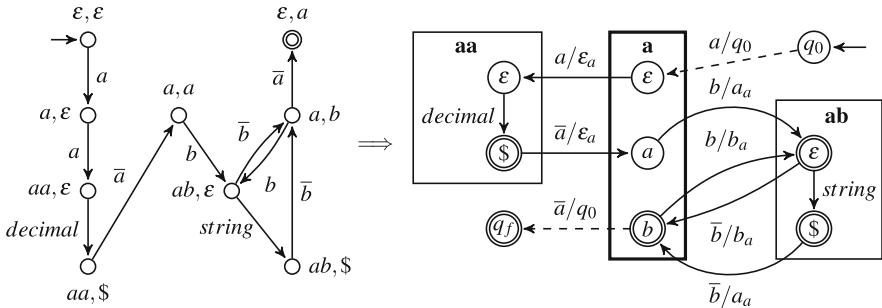


Fig. 12 Similar states in a VPPA are merged and the resulting automaton is translated into an XVPA for stream validation

algorithm converges for a large class of XML in practice when given enough training examples. A detailed description of the algorithm is presented in [103], and an example is shown in Fig. 12. The grammatical inference algorithm is still open for improvements, like incremental learning or probabilistic scenarios, and an experimental evaluation is planned.

Observation Points and Applications

Potential observation points for the presented grammatical inference approach to language-based anomaly detection are in the middleware layer of client-cloud interaction. An anomaly detector learns a specific reference model for an XML-based language that is communicated between a client and a service.

There are two application scenarios in client-cloud interaction with respect to the service interface [104]: If the service interface is statically typed, a learned reference model could help to refine the interface schema, e.g., when it is too general or outdated. In case of a dynamically typed service interface, where no schema for

the interface is available, a learned reference model can detect deviating messages that eventually indicate attack activity.

5 Conclusion

Monitoring of software has many applications, and it is particularly important to show correctness and dependability of services when we accept that software is not fault-free. Faults can lead to security vulnerabilities and service failures that have a tremendous impact on web and cloud services. In this sense, monitoring complements testing and formal methods.

Monitoring observes the behavior of a system to analyze and verify certain properties, and proper observation points that expose the system's state are required. In the cloud computing paradigm, a client loses control over some software components when cloud services are consumed. As the client's capabilities to access or modify software components on the provider-side depend on the particular cloud service and its provider, we distinguish two types of monitoring: black-box and white-box monitoring. In our research we propose a client-controlled middleware approach, where monitoring components in the middleware are fully controlled by the client to deal with black-box software components on the provider-side. The middleware approach restores the client control over consumed cloud services, offering a solution to manage security and quality aspects.

Even though there is a lot of research going on regarding SLAs, in practice, web and cloud services have SLAs that are written in natural language which makes it difficult to monitor the contractually agreed-upon quality of a service. To tackle this problem, research has been done to formally express SLAs such that monitoring of SLA conditions becomes more feasible. SLA monitoring has been primarily provider-centric, i.e., the SLAs are set and monitored by the provider, and it is the client's responsibility to report any violations. A formal specification of SLAs enables negotiation between client and service provider which makes the contracting process more realistic. The formal specification of the SLA and the monitor can be placed on the middleware. Therefore, it allows automated monitoring of agreed-upon service levels on the client-side. The monitoring can also be done on the provider-side and reported to a central component on the middleware for better consistency of the monitoring. One of the challenges in this solution is the synchronization between the different SLA management components on the middleware.

Another monitoring application is to detect intrusions in client-cloud interaction as a security control. We investigate anomaly-based intrusion detection approaches because they are capable of detecting attacks that are not yet known, and we propose language-based anomaly detection for XML-based interaction. Today's web and cloud services use various transport mechanisms to exchange messages for interaction, where XML is a popular semi-structured language to encode messages. An attacker could exploit the client or service by modifying a message,

so unexpected syntax or element content eventually leads to an insecure state when the message is interpreted. We propose an anomaly detection approach, where an algorithm learns a reference model for a specific client-cloud interaction, so deviating messages are eventually detected and properly handled.

Acknowledgements We would like to thank the Christian Doppler Society for supporting this research.

References

1. Abadi, M., Budiu, M., Erlingsson, U., Ligatti, J.: Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.* **13**(1), 1–40 (2009)
2. Aceto, G., Botta, A., de Donato, W., Pescapè, A.: Cloud monitoring: a survey. *Comput. Netw.* **57**(9), 2093–2115 (2013)
3. Ahonen, H.: Generating grammars for structured documents using grammatical inference methods. Tech. Rep. A-1996-4. Department of Computer Science, University of Helsinki (1996)
4. Alonso, G., Casati, F., Kuno, H.A., Machiraj, V.: *Web Services - Concepts, Architectures and Applications*. Springer, Heidelberg (2004)
5. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. *Distrib. Comput.* **2**(3), 117–126 (1987)
6. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC'04*, pp. 202–211. ACM, New York (2004)
7. Alur, R., Madhusudan, P.: Adding nesting structure to words. *J. ACM* **56**(3), 1–43 (2009)
8. Amazon Elastic Compute Cloud: GPU instances. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using_cluster_computing.html (2014). Accessed 09 Sept 2014
9. Amazon Web Services: Amazon web services customer agreement. <http://aws.amazon.com/agreement/> (2008). Accessed 28 Aug 2013
10. Amazon Web Services: Amazon ec2 service level agreement. <http://aws.amazon.com/de/ec2-sla/> (2013). Accessed 20 Nov 2013
11. Android Developers: Sensors overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html (2014). Accessed 09 Sept 2014
12. Apache Commons: BCEL. <http://commons.apache.org/proper/commons-bcel/> (2014). Accessed 10 Sept 2014
13. Ariu, D., Tronci, R., Giacinto, G.: Hmmpayl: an intrusion detection system based on hidden markov models. *Comput. Secur.* **30**(4), 221–241 (2011)
14. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. *Commun. ACM* **53**(4), 50–58 (2010)
15. Avižienis, A., Laprie, J.C.: Dependable computing: from concepts to design diversity. *Proc. IEEE* **74**(5), 629–638 (1986)
16. Avižienis, A., Laprie, J.C., Randell, B.: Dependability and its threats: a taxonomy. In: *Building the Information Society. IFIP International Federation for Information Processing*, vol. 156, pp. 91–120. Springer, New York (2004)
17. Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.* **1**(1), 11–33 (2004)
18. Ayad, A., Dippel, U.: Agent-based monitoring of virtual machines. In: *International Symposium in Information Technology (ITSim)*, pp. 1–6. IEEE, Kuala Lumpur (2010)

19. Barford, P., Kline, J., Plonka, D., Ron, A.: A signal analysis of network traffic anomalies. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement, IMW'02, pp. 71–82. ACM, New York (2002)
20. Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.: Can machine learning be secure? In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS'06, pp. 16–25. ACM, New York (2006)
21. Barros, A., Dumas, M., ter Hofstede, A.H.: Service interaction patterns: towards a reference framework for service-based business process interconnection. Tech. Rep. FIT-TR-2005-02. Faculty of IT, Queensland University of Technology (2005)
22. Bellevue Linux Users Group: The linux information project (linfo). <http://www.linfo.org/index.html> (2007). Accessed 19 Oct 2013
23. Bendrath, R., Mueller, M.: The end of the net as we know it? Deep packet inspection and internet governance. *New Media Soc.* **13**(7), 1142–1160 (2011)
24. Bex, G.J., Neven, F., Van den Bussche, J.: Dtds versus xml schema: a practical study. In: Proceedings of the 7th International Workshop on the Web and Databases, WebDB'04, pp. 79–84. ACM, New York (2004)
25. Bex, G.J., Neven, F., Vansummeren, S.: Inferring xml schema definitions from xml data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB'07, pp. 998–1009. VLDB Endowment, Vienna (2007)
26. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from xml data. *ACM Trans. Web* **4**(4), 1–32 (2010)
27. Bex, G.J., Neven, F., Schwentick, T., Vansummeren, S.: Inference of concise regular expressions and dtds. *ACM Trans. Database Syst.* **35**(2), 1–47 (2010)
28. Bilge, L., Dumitras, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS'12, pp. 833–844. ACM, New York (2012)
29. Binder, W., Hulaas, J., Moret, P.: Advanced java bytecode instrumentation. In: Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java, pp. 135–144. ACM, New York (2007)
30. Boggs, N., Hiremagalore, S., Stavrou, A., Stolfo, S.J.: Cross-domain collaborative anomaly detection: so far yet so close. In: Recent Advances in Intrusion Detection – RAID'11. Lecture Notes of Computer Science, vol. 6961, pp. 142–160. Springer, Heidelberg (2011)
31. Bolzoni, D., Etalle, S., Hartel, P., Zambon, E.: Poseidon: a 2-tier anomaly-based network intrusion detection system. In: 4th IEEE International Workshop on Information Assurance, IWIA'06, pp. 144–156. IEEE, London (2006)
32. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, New York (2003)
33. Bradley, K.A., Lemler, C., Patel, A.C., Lau, R.M.: Time-based monitoring of service level agreements. Cisco Technology, Inc., United States Patent, No. US007082463 B1 (2006)
34. Carpenter, B., Brim, S.: Middleboxes: taxonomy and issues. RFC 3234 (Informational). <http://www.ietf.org/rfc/rfc3234.txt> (2002)
35. Čeleda, P., Krmíček, V.: Flow data collection in large scale networks. In: Advances in IT Early Warning, pp. 30–40. Fraunhofer, Stuttgart (2013)
36. Chan-Tin, E., Heorhiadi, V., Hopper, N., Kim, Y.: The frog-boiling attack: limitations of secure network coordinate systems. *ACM Trans. Inf. Syst. Secur.* **14**(3), 1–23 (2011)
37. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. *ACM Comput. Surv.* **41**(3), 1–58 (2009)
38. Chidlovskii, B.: Schema extraction from xml: a grammatical inference approach. In: Proceedings of the 8th International Workshop on Knowledge Representation Meets Databases, KRDB'01 (2001)
39. Choon, M., Lin, C.Y.J., Wang, X.: A scalable monitoring approach for service level agreements validation. In: International Conference on Network Protocols, ICNP'00, pp. 37–48. IEEE, Osaka (2000)
40. Cisco: Netflow. www.cisco.com/go/netflow. Accessed 18 Oct 2013

41. Comuzzi, M., Kotsokalis, C., Spanoudakis, G., Yahyapour, R.: Establishing and monitoring slas in complex service based systems. In: IEEE International Conference on Web Services, ICWS'09, pp. 783–790. IEEE (2009)
42. Corona, I., Ariu, D., Giacinto, G.: Hmm-web: a framework for the detection of attacks against web applications. In: IEEE International Conference on Communications, ICC'09, pp. 1–6. IEEE, Los Angeles (2009)
43. Criscione, C., Salvaneschi, G., Maggi, F., Zanero, S.: Integrated detection of attacks against browsers, web applications and databases. In: European Conference on Computer Network Defense, EC2ND'09, pp. 37–45. IEEE, Milan (2009)
44. Croll, A., Power, S.: Complete web monitoring: watching your visitors, performance, communities, and competitors. O'Reilly Media, Sebastopol (2009)
45. Curry, E.: Message-oriented middleware. In: Mahmoud, Q.H. (ed.) *Middleware for Communications*. Wiley, Chichester (2005)
46. Dastjerdi, A.V., Tabatabaei, S.G.H., Buyya, R.: A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. *Softw. Pract. Exp.* **42**(4), 501–518 (2012)
47. de la Higuera, C.: *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, Cambridge (2010)
48. Debar, H., Dacier, M., Wespi, A.: Towards a taxonomy of intrusion-detection systems. *Comput. Netw.* **31**(8), 805–822 (1999)
49. Delgado, N., Gates, A., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Trans. Softw. Eng.* **30**(12), 859–872 (2004)
50. Denning, D.E.: An intrusion-detection model. *IEEE Trans. Softw. Eng.* **SE-13**(2), 222–232 (1987)
51. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). <http://www.ietf.org/rfc/rfc5246.txt> (2008). Updated by RFCs 5746, 5878, 6176
52. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W.: Virtuoso: narrowing the semantic gap in virtual machine introspection. In: IEEE Symposium on Security and Privacy, S&P'11, pp. 297–312. IEEE, Washington (2011)
53. Düssel, P., Gehl, C., Laskov, P., Rieck, K.: Incorporation of application layer protocol syntax into anomaly detection. In: *Information Systems Security – ICISS'08*. Lecture Notes of Computer Science, vol. 5352, pp. 188–202. Springer, Heidelberg (2008)
54. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.* **44**(2), 1–42 (2012)
55. Emeakaroha, V.C., Brandic, I., Maurer, M., Dustdar, S.: Low level metrics to high level slas-lom2his framework: bridging the gap between monitored metrics and sla parameters in cloud environments. In: *International Conference on High Performance Computing and Simulation, HPCS'10*, pp. 48–54. IEEE, Caen (2010)
56. Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Buyya, R., De Rose, C.A.: Towards autonomic detection of sla violations in cloud infrastructures. *Futur. Gener. Comput. Syst.* **28**(7), 1017–1029 (2012)
57. Endres-Niggemeyer, B.: The mashup ecosystem. In: *Semantic Mashups*, pp. 1–51. Springer, Heidelberg (2013)
58. Falkenberg, A., Jensen, M., Schwenk, J.: Welcome to ws-attacks.org. <http://www.ws-attacks.org> (2011). Accessed 05 Feb 2013
59. Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee, W., Gong, W.: Anomaly detection using call stack information. In: IEEE Symposium on Security and Privacy, S&P'03, pp. 62–75. IEEE, Washington (2003)
60. Fernau, H.: Learning xml grammars. In: *Machine Learning and Data Mining in Pattern Recognition – MLDM'01*. Lecture Notes of Computer Science, vol. 2123, pp. 73–87. Springer, Heidelberg (2001)
61. Fernau, H.: Identification of function distinguishable languages. *Theor. Comput. Sci.* **290**(3), 1679–1711 (2003)

62. Fielding, R.T.: Rest: architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California (2000)
63. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for unix processes. In: IEEE Symposium on Security and Privacy, S&P'96, pp. 120–128. IEEE, Washington (1996)
64. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic) (2011). <http://www.ietf.org/rfc/rfc6101.txt>
65. Frossi, A., Maggi, F., Rizzo, G., Zanero, S.: Selecting and improving system call models for anomaly detection. In: Detection of Intrusions and Malware, and Vulnerability Assessment – DIMVA'09. Lecture Notes in Computer Science, vol. 5587, pp. 206–223. Springer, Heidelberg (2009)
66. Garfinkel, T.: Traps and pitfalls: practical problems in system call interposition based security tools. In: Proceedings of the Network and Distributed Systems Security Symposium, NDSS'03, pp. 163–176 (2003)
67. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proceedings of the Network and Distributed System Security Symposium, NDSS'03 (2003)
68. Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S., Shim, K.: Xtract: learning document type descriptors from xml document collections. *Data Min. Knowl. Discov.* **7**(1), 23–56 (2003)
69. Garrett, J.J.: Ajax. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications> (2005). Accessed 27 March 2013
70. Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., Radatz, J., Yee, M., Porteous, H., Springsteel, F.: IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. IEEE, Piscataway (1991)
71. Gerhards, R.: The Syslog Protocol. RFC 5424 (Proposed Standard) (2009). <http://www.ietf.org/rfc/rfc5424.txt>
72. Goodloe, A., Pike, L.: Monitoring distributed real-time systems: a survey and future directions. Tech. Rep. NASA/CR-2010-216724. NASA Langley Research Center (2010)
73. Google Developers: Geolocation. <https://developers.google.com/maps/articles/geolocation> (2014). Accessed 09 Sept 2014
74. Görnitz, N., Kloft, M., Rieck, K., Brefeld, U.: Active learning for network intrusion detection. In: Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, AISec'09, pp. 47–54. ACM, New York (2009)
75. Gottschalk, K., Graham, S., Kreger, H., Snell, J.: Introduction to web services architecture. *IBM Syst. J.* **41**(2), 170–177 (2002)
76. Grijzenhout, S., Marx, M.: The quality of the xml web. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM'11, pp. 1719–1724. ACM, New York (2011)
77. Hadžiosmanović, D., Simionato, L., Bolzoni, D., Zambon, E., Etalle, S.: N-gram against the machine: on the feasibility of the n-gram network analysis for binary protocols. In: Research in Attacks, Intrusions, and Defenses – RAID'12. Lecture Notes in Computer Science, vol. 7462, pp. 354–373. Springer, Heidelberg (2012)
78. Handley, M., Paxson, V., Kreibich, C.: Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In: Proceedings of the USENIX Security Symposium, SECURITY'01. USENIX Association (2001)
79. Harrington, D., Presuhn, R., Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (INTERNET STANDARD). <http://www.ietf.org/rfc/rfc3411.txt> (2002). Updated by RFCs 5343, 5590
80. Hauck, R., Reiser, H.: Monitoring of service level agreements with exible and extensible agents. In: Workshop of the OpenView University Association, OVUA'99. Citeseer (1999)
81. Hegewald, J., Naumann, F., Weis, M.: Xstruct: efficient schema extraction from multiple and large xml documents. In: 22nd International Conference on Data Engineering Workshops, ICDEW'06, pp. 81–81. IEEE, Atlanta (2006)

82. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. *J. Comput. Secur.* **6**(3), 151–180 (1998)
83. Hofstede, R., Drago, I., Sperotto, A., Pras, A.: Flow monitoring experiences at the ethernet-layer. In: *Energy-Aware Communications – EUNICE’11. Lecture Notes in Computer Science*, vol. 6955, pp. 134–145. Springer, Heidelberg (2011)
84. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D.: Adversarial machine learning. In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec’11*, pp. 43–58. ACM, New York (2011)
85. Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. *Comput. Netw.* **51**(5), 1239–1255 (2007)
86. Internet Explorer Dev Center: Introduction to the Geolocation API. <http://msdn.microsoft.com/en-us/library/ie/gg589513.aspx> (2014). Accessed 09 Sept 2014
87. iOS Developer Library: CMMotionManager Class Reference. https://developer.apple.com/library/ios/documentation/coremotion/reference/cmmotionmanager_class/Reference/Reference.html (2013). Accessed 09 Sept 2014
88. Jaakkola, H., Thalheim, B.: Exception-aware (information) systems. In: *Information Modelling and Knowledge Bases XXIV. Frontiers in Artificial Intelligence and Applications*, vol. 251, pp. 300–313. IOS Press, Amsterdam (2013)
89. Jayashree, K., Anand, S.: Web service diagnoser model for managing faults in web services. *Comput. Stand. Interfaces* **36**(1), 154–164 (2013)
90. Jensen, M., Gruschka, N., Herkenhöner, R.: A survey of attacks on web services. *Comput. Sci. Res. Dev.* **24**(4), 185–197 (2009)
91. Joshi, K.R., Bunker, G., Jahanian, F., van Moorsel, A., Weinman, J.: Dependability in the cloud: challenges and opportunities. In: *IEEE/IFIP International Conference on Dependable Systems & Networks, 2009, DSN’09*, pp. 103–104. IEEE, Lisbon (2009)
92. Keller, A., Ludwig, H.: IBM research report the WSLA framework: specifying and monitoring service level agreements for web services the WSLA framework: specifying and monitoring. *J. Netw. Syst. Manag.* **11**(1), 57–81 (2003)
93. Kirchner, M.: A framework for detecting anomalies in http traffic using instance-based learning and k-nearest neighbor classification. In: *2nd International Workshop on Security and Communication Networks, IWSCN’10*, pp. 1–8. IEEE, Karlstad (2010)
94. Ko, C., Fink, G., Levitt, K.: Automated detection of vulnerabilities in privileged programs by execution monitoring. In: *10th Annual Computer Security Applications Conference, ACSAC’94*, pp. 134–144. IEEE, Orlando (1994)
95. Ko, C., Ruschitzka, M., Levitt, K.: Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In: *IEEE Symposium on Security and Privacy, S&P’97*, pp. 175–187. IEEE, Oakland (1997)
96. Kosala, R., Blockeel, H., Bruynooghe, M., Van den Bussche, J.: Information extraction from structured documents using k-testable tree automaton inference. *Data Knowl. Eng.* **58**(2), 129–158 (2006)
97. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: *Proceedings of the 10th ACM Conference on Computer and Communication Security, CCS’03*, pp. 251–261. ACM, New York (2003)
98. Krüger, T., Gehl, C., Rieck, K., Laskov, P.: Tokdoc: a self-healing web application firewall. In: *Proceedings of the 2010 ACM Symposium on Applied Computing, SAC’10*, pp. 1846–1853. ACM, New York (2010)
99. Krüger, T., Krämer, N., Rieck, K.: Asap: automatic semantics-aware analysis of network payloads. In: *Privacy and Security Issues in Data Mining and Machine Learning – PSDML’10. Lecture Notes of Computer Science*, vol. 6549, pp. 50–63. Springer, Heidelberg (2011)
100. Kumar, V., Madhusudan, P., Viswanathan, M.: Minimization, learning, and conformance testing of boolean programs. In: *CONCUR 2006 – Concurrency Theory. Lecture Notes of Computer Science*, vol. 4137, pp. 203–217. Springer, Heidelberg (2006)

101. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming xml. In: Proceedings of the 16th International Conference on World Wide Web, WWW'07, pp. 1053–1062. ACM, New York (2007)
102. Lamanna, D.D., Skene, J., Emmerich, W.: Slang: a language for service level agreements. In: Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems, FTDCS'03, pp. 100–106. IEEE, Washington (2003)
103. Lampesberger, H.: A grammatical inference approach to language-based anomaly detection in xml. In: 2013 International Conference on Availability, Reliability and Security, ECTCM'13 Workshop, pp. 685–693. IEEE, Washington (2013)
104. Lampesberger, H.: Technologies for Web and cloud service interaction: a survey. *Serv. Oriented Comput. Appl.* (2015) doi: [10.1007/s11761-015-0174-12015](https://doi.org/10.1007/s11761-015-0174-12015)
105. Lampesberger, H., Winter, P., Zeilinger, M., Hermann, E.: An on-line learning statistical model to detect malicious web requests. In: Security and Privacy in Communication Networks. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 96, pp. 19–38. Springer, Heidelberg (2012)
106. Lampesberger, H., Zeilinger, M., Hermann, E.: Statistical modeling of web requests for anomaly detection in web applications. In: Advances in IT Early Warning, pp. 91–101. Fraunhofer AISEC, Garching (2013)
107. Lamport, L.: Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.* **SE-3**(2), 125–143 (1977)
108. Lazarevic, A., Kumar, V., Srivastava, J.: Intrusion detection: a survey. In: Managing Cyber Threats. Massive Computing, vol. 5, pp. 19–78. Springer, New York (2005)
109. ldv_alt: Project page: strace. Online. <http://freecode.com/projects/strace>. Accessed 18 Oct 2013
110. Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L.: SOCKS Protocol Version 5. RFC 1928 (Proposed Standard) (1996). <http://www.ietf.org/rfc/rfc1928.txt>
111. Leucker, M., Schallhart, C.: A brief account of runtime verification. *J. Logic Algebraic Program.* **78**(5), 293–303 (2009)
112. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement WSLA Language Specification. IBM Corporation, pp. 815–824 (2003)
113. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
114. Magazinius, J., Russo, A., Sabelfeld, A.: On-the-fly inlining of dynamic security monitors. *Comput. Secur.* **31**(7), 827–843 (2012)
115. Magazinius, J., Hedlin, D., Sabelfeld, A.: Architectures for inlining security monitors in web applications. In: International Symposium on Engineering Secure Software and Systems, ESSoS'14. Springer, Heidelberg (2014)
116. Maggi, F., Robertson, W., Kruegel, C., Vigna, G.: Protecting a moving target: addressing web application concept drift. In: Recent Advances in Intrusion Detection – RAID'09. Lecture Notes of Computer Science, vol. 5758, pp. 21–40. Springer, Heidelberg (2009)
117. Maggi, F., Matteucci, M., Zanero, S.: Detecting intrusions through system call sequence and argument analysis. *IEEE Trans. Dependable Secure Comput.* **7**(4), 381–395 (2010)
118. Maggi, F., Zanero, S.: Is the future web more insecure? Distractions and solutions of new-old security issues and measures. In: 2nd Worldwide Cybersecurity Summit, WCS'11, pp. 1–9. IEEE, London (2011)
119. Mahoney, M.V.: Network traffic anomaly detection based on packet bytes. In: Proceedings of the 2003 ACM Symposium on Applied computing, SAC'03, pp. 346–350. ACM, New York (2003)
120. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'02, pp. 376–385. ACM, New York (2002)
121. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.* **31**(3), 770–813 (2006)
122. Michael, C.C., Ghosh, A.: Simple, state-based approaches to program-based anomaly detection. *ACM Trans. Inf. Syst. Secur.* **5**(3), 203–237 (2002)

123. Mlýnková, I.: An analysis of approaches to XML schema inference. In: IEEE International Conference on Signal Image Technology and Internet Based Systems, SITIS'08, pp. 16–23. IEEE, Bali (2008)
124. Mlýnková, I., Nečaský, M.: Towards inference of more realistic xsds. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC'09, pp. 639–646. ACM, New York (2009)
125. Molina-Jimenez, C., Shrivastava, S., Crowcroft, J., Gevros, P.: On the monitoring of contractual service level agreements. In: 1st IEEE International Workshop on Electronic Contracting, WEC'04, pp. 1–8. IEEE, San Diego (2004)
126. Mooney, J.D.: Bringing portability to the software process. Department of Statistics and Computer Science, West Virginia University, Morgantown (1997)
127. Murata, M.: Relax ng. <http://relaxng.org/> (2013). Accessed 01 Feb 2013
128. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of xml schema languages using formal language theory. *ACM Trans. Internet Technol.* **5**(4), 660–704 (2005)
129. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous system call detection. *ACM Trans. Inf. Syst. Secur.* **9**(1), 61–93 (2006)
130. Nance, K., Bishop, M., Hay, B.: Virtual machine introspection: observation or interference? *IEEE Secur. Privacy Mag.* **6**(5), 32–37 (2008)
131. Necula, G.C., McPeak, S., Rahul, S., Weimer, W.: Cil: Intermediate language and tools for analysis and transformation of c programs. In: *Compiler Construction. Lecture Notes in Computer Science*, vol. 2304, pp. 213–228. Springer, Heidelberg (2002)
132. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.* **42**(6), 89–100 (2007)
133. Niemi, O.P., Levomäki, A., Manner, J.: Dismantling intrusion prevention systems. *ACM SIGCOMM Comput. Commun. Rev.* **42**(4), 285–286 (2012)
134. Nusayr, A., Cook, J.: Extending AOP to support broad runtime monitoring needs. In: *Conference on Software Engineering and Knowledge Engineering*, pp. 438–441 (2009)
135. Nusayr, A., Cook, J.: Using aop for detailed runtime monitoring instrumentation. In: *Proceedings of the Seventh International Workshop on Dynamic Analysis, WODA'09*, pp. 8–14. ACM, New York (2009)
136. OpenSuSe Documentation: Understanding linux audit. http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html. Accessed 18 Oct 2013
137. Oracle: Solaris dynamic tracing guide. <http://docs.oracle.com/cd/E19253-01/817-6223/>. Accessed 18 Oct 2013
138. Parameswaran, A., Chaddha, A.: Cloud interoperability and standardization. *SETLabs Brief.* **7**(7), 19–26 (2009)
139. Pautasso, C., Zimmermann, O., Leymann, F.: Restful web services vs. “big” web services: making the right architectural decision. In: *Proceedings of the 17th International Conference on World Wide Web, WWW'08*, pp. 805–814. ACM, New York (2008)
140. Paxson, V.: Bro: A system for detecting network intruders in real-time. *Comput. Netw.* **31**(23–24), 2435–2463 (1999)
141. Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., Lee, W.: Mcpad: a multiple classifier system for accurate payload-based anomaly detection. *Comput. Netw.* **53**(6), 864–881 (2009)
142. Picalausa, F., Servais, F., Zimányi, E.: Xevolve: an XML schema evolution framework. In: *Proceedings of the 2011 ACM Symposium on Applied Computing, SAC'11*, pp. 1645–1650. ACM, New York (2011)
143. Plattner, B., Nievergelt, J.: Monitoring program execution: a survey. *Computer* **14**(11), 76–93 (1981)
144. Ptacek, T.H., Newsham, T.N.: Insertion, evasion, and denial of service: eluding network intrusion detection. Tech. rep., Secure Networks, Inc. http://insecure.org/stf/secnet_ids/secnet_ids.html (1998). Accessed 13 Oct 2013
145. Rady, M.: Parameters for service level agreements generation in cloud computing a client-centric vision. In: *Advances in Conceptual Modeling – CMS'12. Lecture Notes of Computer Science*, vol. 7518, pp. 13–22. Springer, Heidelberg (2012)

146. Rady, M.: Generating an excerpt of a service level agreement from a formal definition of non-functional aspects using owl. *J. Univers. Comput. Sci.* **20**(3), 366–384 (2014)
147. Raeymaekers, S., Bruynoghe, M., den Bussche, J.: Learning (k, l)-contextual tree languages for information extraction from web pages. *Mach. Learn.* **71**(2), 155–183 (2008)
148. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard). <http://www.ietf.org/rfc/rfc6347.txt> (2012)
149. Richters, M., Gogolla, M.: Aspect-oriented monitoring of uml and ocl constraints. In: AOSD Modeling With UML Workshop, 6th International Conference on the Unified Modeling Language (UML) (2003)
150. Rieck, K.: Machine learning for application-layer intrusion detection. Ph.D. thesis, Berlin Institute of Technology, TU Berlin (2009)
151. Robertson, W., Vigna, G., Kruegel, C., Kemmerer, R.: Using generalization and characterization techniques in the anomaly-based detection of web attacks. In: Proceedings of the Network and Distributed System Security Symposium, NDSS'06 (2006)
152. Robertson, W., Maggi, F., Kruegel, C., Vigna, G.: Effective anomaly detection with scarce training data. In: Proceedings of the Network and Distributed System Security Symposium, NDSS'10 (2010)
153. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration, LISA'99, pp. 229–238. USENIX Association, Seattle (1999)
154. Romano, L., De Mari, D., Jerzak, Z., Fetzer, C.: A novel approach to qos monitoring in the cloud. In: 1st International Conference on Data Compression, Communications and Processing, CCP'11, pp. 45–51. IEEE, Palinuro (2011)
155. Rosenberg, F., Platzer, C., Dustdar, S.: Bootstrapping performance and dependability attributes of web services. In: International Conference on Web Services, ICWS'06, pp. 205–212. IEEE, Chicago (2006)
156. Rubinstein, B.I., Nelson, B., Huang, L., Joseph, A.D., Lau, S.h., Rao, S., Taft, N., Tygar, J.D.: Antidote: understanding and defending against poisoning of anomaly detectors. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, IMC'09, pp. 1–14. ACM, New York (2009)
157. Sabelfeld, A., Myers, A.: Language-based information-flow security. *IEEE J. Select. Areas Commun.* **21**(1), 5–19 (2003)
158. Sahai, A., Machiraju, V., Sayal, M., Moorsel, A., Casati, F.: Automated sla monitoring for web services. In: Management Technologies for E-Commerce and E-Business Applications – DSOM'02. Lecture Notes in Computer Science, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
159. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. *ACM Comput. Surv.* **42**(3), 1–42 (2010)
160. Sandhu, R., Samarati, P.: Access control: principle and practice. *IEEE Commun. Mag.* **32**(9), 40–48 (1994)
161. SAP: Message Flow Monitoring. http://docs.oracle.com/cd/E21764_01/core.1111/e10043/audintro.htm (2011). Accessed 11 Sept 2014
162. Sassaman, L., Patterson, M., Bratus, S., Locasto, M.: Security applications of formal language theory. *IEEE Syst. J.* **7**(3), 489–500 (2013)
163. Schewe, K.D., Bósa, K., Lampesberger, H., Ma, J., Rady, M., Vleju, M.B.: Challenges in cloud computing. *Scalable Comput. Pract. Exp.* **12**(4), 385–390 (2011)
164. Schewe, K.D., Thalheim, B., Wang, Q.: Updates, schema updates and validation of xml documents - using abstract state machines with automata-defined states. *J. Univers. Comput. Sci.* **15**(10), 2028–2057 (2009)
165. Schneider, F.B.: Enforceable security policies. *ACM Trans. Inf. Syst. Secur.* **3**(1), 30–50 (2000)
166. Schroeder, B.: On-line monitoring: a tutorial. *Computer* **28**(6), 72–78 (1995)
167. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: Proceedings of the 21st ACM Symposium on Principles of Database Systems, PODS'02, pp. 53–64. ACM, New York (2002)

168. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: IEEE Symposium on Security and Privacy, S&P'01, pp. 144–155. IEEE, Washington (2001)
169. Shackel, B.: Usability-context, framework, definition, design and evaluation. In: Human Factors for Informatics Usability, pp. 21–37. Cambridge University Press, Cambridge (1991)
170. Somayaji, A., Forrest, S.: Automated response using system-call delays. In: Proceedings of the 9th USENIX Security Symposium, SECURITY'00 (2000)
171. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy, pp. 305–316 (2010)
172. Song, Y., Locasto, M.E., Stavrou, A., Keromytis, A.D., Stolfo, S.J.: On the infeasibility of modeling polymorphic shellcode. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS'07, pp. 541–551. ACM, New York (2007)
173. Song, Y., Keromytis, A., Stolfo, S.J.: Spectrogram: a mixture-of-markov-chains model for anomaly detection in web traffic. In: Proceedings of the Network and Distributed System Security Symposium, NDSS'09 (2009)
174. Soyulu, A., Mödritscher, F., Wild, F., Causmaecker, P.D., Desmet, P.: Mashups by orchestration and widget-based personal environments: key challenges, solution strategies, and an application. *Program Electron. Libr. Inf. Syst.* **46**(4), 383–428 (2012)
175. Spring, J.: Monitoring cloud computing by layer, part 1. *IEEE Secur. Privacy Mag.* **9**(2), 66–68 (2011)
176. Spring, J.: Monitoring cloud computing by layer, part 2. *IEEE Secur. Privacy Mag.* **9**(3), 52–55 (2011)
177. Stevens, W.R.: *TCP/IP Illustrated: The Protocols*, vol. 1. Addison-Wesley, Boston (1993)
178. Thalheim, B.: Towards a theory of conceptual modelling. *J. Univers. Comput. Sci.* **16**(20), 3102–3137 (2010)
179. The Apache Software Foundation: Apache module mod_proxy. http://httpd.apache.org/docs/2.0/mod/mod_proxy.html (2013). Accessed 18 Nov 2013
180. The Network Encyclopedia: Circuit level gateway. <http://www.thenetworkencyclopedia.com/entry/circuit-level-gateway/> (2013). Accessed 15 Sept 2014
181. The SAX Project: Simple api for xml (sax). <http://www.saxproject.org/> (2004). Accessed 24 Jan 2013
182. Thottan, M., Ji, C.: Anomaly detection in ip networks. *IEEE Trans. Signal Process.* **51**(8), 2191–2204 (2003)
183. TrustedBSD Project: Openbsm: Open source basic security module (bsm) audit implementation. <http://www.trustedbsd.org/openbsm.html>. Accessed 18 Oct 2013
184. Valdes, A., Skinner, K.: Adaptive, model-based monitoring for cyber attack detection. In: *Recent Advances in Intrusion Detection – RAID'00*. Lecture Notes in Computer Science, vol. 1907, pp. 80–93. Springer, Heidelberg (2000)
185. W3C: Web Services Addressing (WS-Addressing). <http://www.w3.org/Submission/ws-addressing/> (2004). Accessed 03 March 2014
186. W3C: Document object model (dom). <http://www.w3.org/DOM/> (2005). Accessed 24 Jan 2013
187. W3C: SOAP Version 1.2 Part 1: Messaging Framework, 2nd edn. <http://www.w3.org/TR/soap12-part1/> (2007). Accessed 20 Feb 2014
188. W3C: XML Schema. <http://www.w3.org/XML/Schema.html> (2010). Accessed 11 Feb 2013
189. W3C: XML Schema Part 2: Datatypes, 2nd edn. <http://www.w3.org/TR/xmlschema11-2/> (2012). Accessed 22 March 2013
190. Wagner, D., Dean, R.: Intrusion detection via static analysis. In: IEEE Symposium on Security and Privacy, S&P'01, pp. 156–168. IEEE, Washington (2001)
191. Wagner, D., Soto, P.: Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS'02, pp. 255–264. ACM, New York (2002)
192. Wang, J., Bigham, J.: Anomaly detection in the case of message oriented middleware. In: Proceedings of the 2008 Workshop on Middleware Security, MidSec'08, pp. 40–42. ACM, New York (2008)

193. Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: *Recent Advances in Intrusion Detection – RAID’04*. Lecture Notes of Computer Science, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)
194. Wang, K., Parekh, J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: *Recent Advances in Intrusion Detection – RAID’06*. Lecture Notes of Computer Science, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
195. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. *IEEE Netw.* **24**(4), 19–24 (2010)
196. WebSphere Software: Introduction to Oracle Fusion Middleware Audit Framework. http://docs.oracle.com/cd/E21764_01/core.1111/e10043/audintro.htm (2011). Accessed 11 Sept 2014
197. WebSphere Software: Using WebSphere Message Broker log and trace files. http://publib.boulder.ibm.com/infocenter/wtxdoc/v8r2m0/index.jsp?topic=/com.ibm.websphere.dtx.wtx4wmb.doc/references/r_wtx4wmb_using_wmb_log_and_trace_files.htm (2014). Accessed 11 Sept 2014
198. Wieder, P., Butler, J.M., Theilmann, W., Yahyapour, R.: *Service Level Agreements for Cloud Computing*. Springer, New York (2011)
199. Winter, P., Lampesberger, H., Zeilinger, M., Hermann, E.: On detecting abrupt changes in network entropy time series. In: *Communications and Multimedia Security – CMS’11*. Lecture Notes of Computer Science, vol. 7025, pp. 194–205. Springer, Heidelberg (2011)
200. Wojtczuk, R.: Libnids. <http://libnids.sourceforge.net/> (2010). Accessed 01 Nov 2013
201. Xie, Y., Yu, S.Z.: A dynamic anomaly detection model for web user behavior based on hsmm. In: *10th International Conference on Computer Supported Cooperative Work in Design, CSCWD’06*, pp. 1–6. IEEE, Nanjing (2006)
202. Xie, Y., Yu, S.Z.: A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Trans. Netw.* **17**(1), 54–65 (2009)
203. Zanero, S., Savaresi, S.M.: Unsupervised learning techniques for an intrusion detection system. In: *Proceedings of the 2004 ACM Symposium on Applied Computing, SAC’04*, pp. 412–419. ACM, New York (2004)
204. Zhou, J., Gollman, D.: A fair non-repudiation protocol. In: *IEEE Symposium on Security and Privacy, S&P’96*, pp. 55–61. IEEE, Washington (1996)

Formal Reliability Models for Web Services

Raffaella Mirandola, Pasqualina Potena, Elvinia Riccobene,
and Patrizia Scandurra

Abstract In Web services (WS), software applications are dynamically built by assembling over a network existing, loosely coupled, distributed, and heterogeneous services. Reliability is one of the most important quality dimensions for Web services, since predicting their reliability is fundamental to appropriately drive the selection and the assembly of services. This chapter presents two approaches to predict the reliability of a Web service architecture. The first one is based on the Business Process Execution Language (BPEL), the de facto standard executable language for specifying actions within business processes with Web services. The second one is based on the SCA-ASM, a lightweight formal language for modeling service-oriented applications, which is based on the OASIS (Organization for the Advancement of Structured Information Standards) standard Service Component Architecture for heterogeneous service assembly and on the formal method abstract state machines (ASMs) for modeling service behavior, interactions, and orchestration in an abstract but executable way. Through a set of experimental results, we show how the two models work on a smartphone mobile application example, and we discuss the effectiveness of the SCA-ASM approach in comparison with the BPEL-based approach.

R. Mirandola
Politecnico di Milano, Milano, Italy
e-mail: raffaella.mirandola@polimi.it

P. Potena • P. Scandurra (✉)
Università degli Studi di Bergamo, Dalmine (BG), Italy
e-mail: pasqualina.potena@unibg.it; patrizia.scandurra@unibg.it

E. Riccobene
Università degli Studi di Milano, Crema, Italy
e-mail: elvinia.riccobene@unimi.it

1 Introduction

Web service (WS) systems are service-oriented systems where computing software applications are dynamically built by assembling over a network existing, loosely coupled, distributed, and heterogeneous services.

It has been widely recognized [11, 13, 40] that the prediction of nonfunctional properties of these systems is a crucial design-time concern. Architectural decisions, indeed, including selection of services and the structure of the workflow, may significantly affect the qualities of the resulting system, such as their reliability, performance, or cost. This chapter focuses on *reliability*. Early assessment of reliability is one of the challenges of Web service architectures and a key factor to developing dependable software.

In the literature, different procedures exist for system reliability prediction based on different assumptions and applicable at different granularities of information [13, 21, 24, 26, 28]. These techniques can be applied for several purposes such as to evaluate design feasibility, to compare design alternatives, to assist in evaluating the significance of reported failures, to trade off system design factors, to track reliability improvement, to appropriately allocate validation/testing effort, and to identify potential failure areas and maintain an acceptable reliability level under environmental extremes.

Dealing with WS makes the software reliability evaluation more difficult. WS, indeed, are owned (developed, deployed, maintained, and operated) by different stakeholders or providers and the way the orchestration is carried out cannot be foreseen at the time the composed WS is specified. It will depend on several aspects, such as the availability of the involved services, the services' responses, the network status, and on unexpected error conditions. The early assessment of the reliability and availability of a service composition is thus a key challenge of service architectures and a key factor when implementing dependable software.

This work proposes two reliability prediction methods: one based on the Web Services Business Process Execution Language (WS-BPEL or BPEL for short) [3] and the other based on the service-oriented component model SCA-ASM [31, 35–37]. In literature some papers tackled the problem of composing a service-oriented system from publicly available Web services (e.g., [45]), taking into account different types of Web service failures. Several approaches use notations based on BPEL, which is the de facto standard executable language for specifying actions within business processes with Web services (see, e.g., [8, 43] and [12]).

On the other side, the SCA-ASM has been recently proposed as a lightweight formal language for modeling both architecture and behavior aspects of a service application. This component model is based on the OASIS open standard *Service Component Architecture* (SCA) [33] for heterogeneous service assembly, and on the formal method *Abstract State Machines* (ASMs) [6] for modeling notions of

service behavior, interactions, orchestration, and fault and compensation handling in an abstract but executable way. Since the SCA-ASM relies on the SCA design framework, it is supported by the runtime environment Tuscany [4], thus simplifying the prototyping, analysis, development, and deployment of service compositions.

The SCA-ASM reliability model [31, 37] exploits ideas taken from *architecture-based* and *path-based* reliability models [24]. It is based on a reliability model of an SCA-ASM component by considering failures specific to the nature of the ASMs and allows computing, in an automatic and compositional way, the reliability of an SCA assembly involving SCA-ASM components.

Besides to be compositional and applicable at design phase as well as at runtime, there are some other potential advantages of the second approach. It relies on a unique SCA-ASM component model which is both the “design-oriented model” of the component assembly and the “formal analysis-oriented model” leading the reliability analysis. With regard to other classical approaches that tie architectural models (or flavors of Unified Modeling Language (UML) and other modeling notations) to formal reliability models such as Markov or Bayesian models (see related works in Sect. 8), the SCA-ASM reliability model combines the reliability prediction of the service orchestrator with those of other service components; this leads to a more accurate estimation of the reliability.

In this chapter, we present a direct comparison between a BPEL-based model and the SCA-ASM reliability model. Through a case study, we show the behavior of the two reliability models. Specifically, when considering the same modeling abstraction level, the two reliability models are equivalent. However, since the SCA-ASM reliability model allows a more detailed modeling and computation of the reliability of a single Web component, the reliability estimation of the overall Web service system with SCA-ASM provides more accurate results than the ones obtained with the BPEL-based model. Indeed, in the BPEL-based approach, the reliability of a single Web component can be given somehow by external evaluation. The comparison suggests that further benefit can be obtained by combining the use of the two models. Indeed, while the BPEL-based reliability model can be used for the evaluation of the whole process, the SCA-ASM reliability model can be used to evaluate the reliability of the single services, thus providing more accurate results than the one obtained with the current BPEL-based reliability models.

The chapter is organized as follows. Section 2 recalls some basic concepts concerning reliability prediction and makes precise some assumptions we make on our reliability models. Section 3 presents a smartphone mobile application as the running example of this chapter. Sections 4.1 and 4.2 give some background on the BPEL and the SCA-ASM modeling languages. Their reliability models are presented, respectively, in Sects. 5 and 6. Section 7 presents the results of the comparison between the two reliability models. Section 8 describes some related work. Finally, Sect. 9 concludes the chapter and sketches some future research directions.

2 Reliability Prediction Basics

Reliability is one of the major factors of software quality and is defined as the “probability of failure-free software operation for a specified period of time in a specified environment” [41]. *Reliability prediction* is a common form of reliability analysis to predict the failure rate of components and the overall system reliability. Reliability predictions are useful to evaluate design feasibility, compare design alternatives, identify potential failure areas, trade off system design factors, and track reliability improvement. A reliability prediction can also assist in evaluating the significance of reported failures, and it can be used to maintain an acceptable reliability level under environmental extremes. Reliability strongly depends on two main concerns. First, the reliability of a software system depends on the reliability of individual components, component interactions, and the execution environment. Second, reliability depends on how the system will be used (*usage profile* or *operational profile*). Since reliability (like availability) is an execution quality, the impact of faults on reliability differs depending on how the system is used, i.e., how often the faulty part of the system is executed. The analysis of different ways and frequencies to execute the system is a challenge to reliability prediction, especially when the usage profiles are unknown beforehand.

In the last few years, many reliability prediction methods for software that is assembled from basic elements (e.g., objects, components, or services) have been introduced [21, 24, 28]. Basically, the existing techniques can be classified in *path-based models* and *state-based models* [24]. The former ones represent the system architecture as a combination of the possible execution paths, whereas the latter ones as a combination of the possible states of the system.

Possible failures occurring during the execution of a Web application can be classified as *crash failures* that provoke the irreversibly crash of the whole system and *no-crash failures* that do not provoke the immediate termination of the whole system but that manifest themselves by returning an erroneous message. The reliability models we present both consider only *crash failures* and assume that a failure of a service in the Web service composition provokes the failure of the whole application.¹

There are some assumptions underlying our reliability models. Most of them are common to many existing reliability approaches (see, e.g., the surveys [28, 30]) and are necessary to be able to provide in an efficient way analytical results that, even if approximate with respect to the more complex reality, can give meaningful insights to system designers. (1) The components communicate by exchanging synchronous

¹Such assumption is not too restrictive. It is a common practice in many reliability modeling approaches (see, e.g., the survey [28]).

messages. (2) The components' failures are independent of one another. We assume that a component's failure provokes the crash of the whole system, namely, the system straightforwardly stops its execution. The inclusion in our model of different types of failures and of error propagation analysis is at present under study. (3) The model parameters' uncertainties [15] are not dealt since this kind of sensitivity analysis is out of the scope of this work.

3 Running Example: A Multimedia Service Application

This section presents a smartphone application used throughout the chapter to exemplify our approach. The example application is inspired by the Web service composition example used in [34] and originally presented in [2]. Readers interested in the application details that we do not provide here can refer to [34]. Figure 1 (taken from [2]) shows the multimedia delivery scenario. The application called *Multimedia Service App* provides an end-user multimedia service to subscribed users. The news includes text and topical videos available in MPEG 2 format. The news provider requires additional services to serve the user's request: a transcoding service for the multimedia content to fit the target format, a compression service to adapt the content to the wireless link, a text translation service for the news ticker, and also a merging service to integrate the ticker with the video stream for the limited smartphone display. Besides, we assume that the smartphone user can require also a geographical map showing its location [2]. Precisely, in this work, we will consider the following three functionalities: (a) require news in textual format, (b) require news with both textual and video content, and (c) provide a geographical map with user location.

4 Background Concepts

This section provides some basic concepts on BPEL and SCA-ASM useful to understand and compare the reliability models for (Web) service-oriented applications presented in Sects. 5 and 6, respectively.

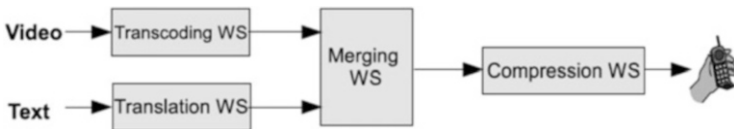


Fig. 1 Multimedia service scenario

Table 1 BPEL-structured activities

Name	Description
<i>Sequence</i>	Sequential execution of activities
<i>Switch</i>	Conditional execution of activities
<i>While</i>	Repeated execution of activities in a loop
<i>Flow</i>	Concurrent execution of activities

4.1 BPEL Composition of Interacting Web Services

From a service-oriented perspective, a business process is a means to have services interact to specify specific requests. The service orchestration logic can be expressed in BPEL [3]. We adopt a general definition of *software service*: it is a self-contained deployable software module containing data and operations, which provides/requires services to/from other elementary elements. A *service instance* is a specific implementation of a service.

As in [12], we here refer to a significant subset of the whole BPEL definition. Specifically, besides the primitive “invoke activity,” which specifies the synchronous or asynchronous invocation of a Web service, we consider the kinds of *structured activities* described in Table 1. A detailed description of BPEL is out of the scope of this chapter.

Running Example As an example of BPEL composition, Fig. 2 sketches the BPEL code² of a flow activity for the orchestration of four elementary services of the *Multimedia Service App* presented in Sect. 3. It corresponds to the functionality (1) to provide the news ticker in textual format only. Besides the normal control flow of service invocations for performing request-response interactions, a conditional branch introduces a decision point in case of failure of the text translation. If the text translation fails, a fake news ticker is returned directly from the service component *Compression* to the service component *Client*. Note that the service component *Client* processes information on the client’s behalf.

4.2 SCA-ASM Modeling Language

The SCA-ASM Modeling Language [35, 36] complements the SCA component model with the ASM model of computation to provide ASM-based formal and executable descriptions of the services *internal behavior*, *orchestration*, and *interactions*. According to this implementation type, a service-oriented component is

²BPEL defines business processes using an XML-based language. There is no standard graphical notation for BPEL. Some vendors have invented their own notations. Consider the standard Business Process Model and Notation (BPMN) [10] as a graphical front end to capture BPEL process descriptions.

Fig. 2 BPEL code for the service “Requiring news in textual format”

```

...
<sequence>
  <invoke ...
    service name = "Client" .../>
  <invoke ...
    service name = "Multimedia" .../>
  <invoke ...
    service name = "Client" .../>
  <invoke ...
    service name = "Translation" .../>
  <switch ...>
    <case condition= "TextTranslation=OK">
      <sequence>
        <invoke ...
          service name = "Multimedia" .../>
        <invoke ...
          service name = "Compression" .../>
        <invoke ...
          service name = "Client" .../>
        <invoke ...
          service name = "Multimedia" .../>
      </sequence>
    </case>
    <otherwise>
      <sequence>
        <invoke ...
          service name = "Compression" .../>
        <invoke ...
          service name = "Multimedia" .../>
        <invoke ...
          service name = "Client" .../>
        <invoke ...
          service name = "Multimedia" .../>
      </sequence>
    </otherwise>
  </switch>
</sequence>

```

an ASM endowed with (at least) one agent (a business partner or role) able to be engaged in conversational interactions with other agents by providing and requiring services to/from other service-oriented components’ agents. The service behaviors encapsulated in an SCA-ASM component are captured by ASM transition rules. We assume the reader to be familiar with the ASM formalism.

Figure 3 shows the shape of an SCA-ASM component A and the corresponding ASM modules for the provided interface AService (on the left) and the skeleton of the component itself (on the right) using the textual notation ASMETA/AsmetaL³

³Two grammatical conventions must be recalled: a variable identifier starts with \$ and a rule identifier begins with “r_.”

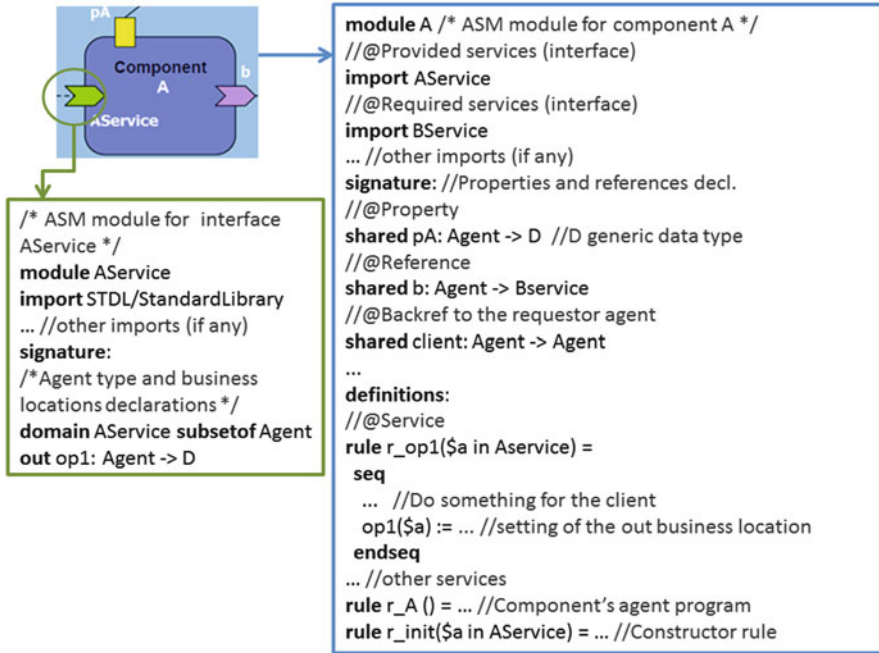


Fig. 3 SCA-ASM component shape

[5, 42] and the @annotations to denote SCA concepts (i.e., references, properties, etc.).

ASM rule constructors and predefined ASM rules (i.e., named ASM rules made available as model library) are used as basic SCA-ASM behavioral primitives. They are recalled in Table 2 by separating them according to the separation of concerns *computation*, *communication*, and *coordination*. In particular, communication primitives provide both synchronous and asynchronous interaction styles (corresponding, respectively, to the *request-response* and *one-way* interaction patterns of the SCA standard). Communication relies on an *abstract message-passing* mechanism by adopting the default SCA binding (`binding.sca`) for message delivering. SCA-ASM rule constructors can be combined to model specific interaction and orchestration patterns in well-structured and modularized entities.

Currently, the implementation scope of an SCA-ASM component is *composite*, i.e., a single component instance (a single ASM) is created for all service calls of the component. The other two SCA implementation scopes, *stateless* (to create a new component instance for each service call) and *conversation* (to create a component instance for each conversation), are not yet supported.

SCA-ASM modeling constructs for fault/compensation handling are also supported (see [35, 36]), but are not reported here since they are related to *fault*

Table 2 SCA-ASM rule constructors

<i>Computation and coordination</i>	
<i>Skip rule</i>	skip do nothing
<i>Update rule</i>	$f(t_1, \dots, t_n) := t$ update the value of f at t_1, \dots, t_n to t
<i>Call rule</i>	$R[x_1, \dots, x_n]$ call rule R with parameters x_1, \dots, x_n
<i>Let rule</i>	let $x = t$ in R assign the value of t to x and then execute R
<i>Conditional rule</i>	if ϕ then R_1 else R_2 endif if ϕ is true, then execute rule R_1 , otherwise R_2
<i>Iterate rule</i>	while ϕ do R execute rule R until ϕ is true
<i>Seq rule</i>	seq $R_1 \dots R_n$ endseq rules $R_1 \dots R_n$ are executed in sequence without exposing intermediate updates
<i>Parallel rule</i>	par $R_1 \dots R_n$ endpar rules $R_1 \dots R_n$ are executed in parallel
<i>Forall rule</i>	forall x with ϕ do $R(x)$ forall x satisfying ϕ execute R
<i>Choose rule</i>	choose x with ϕ do $R(x)$ choose an x satisfying ϕ and then execute R
<i>Split rule</i>	forall $n \in N$ do $R(n)$ split N times the execution of R
<i>Spawn rule</i>	spawn child with R create a child agent with program R
<i>Communication</i>	
<i>Send rule</i>	wsend[lnk, R, snd] send data snd to lnk in reference to rule R (no blocking, no acknowledgment)
<i>Receive rule</i>	wreceive[lnk, R, rcv] receive data rcv from lnk in reference to R (blocks until data are received, no ack)
<i>SendReceive rule</i>	wsendreceive[lnk, R, snd, rcv] send data snd to lnk in reference to R waits for data rcv to be sent back (no ack)
<i>Reply rule</i>	wreply[lnk, R, snd] returns data snd to lnk , as response of R request received from lnk (no ack)

tolerance concepts that we do not take into account in the reliability model presented here.

An SCA-ASM design environment [9, 35] was developed by integrating the Eclipse-based SCA Composite Designer, the SCA runtime platform Tuscany [4], and the simulator ASMETA/AsmetaS [5, 22, 42].

Running Example Figure 4 shows the software architecture of the running example using SCA. It is a thin client/server application: the server part is the real application hosted, for example, on a cloud, while the client (not shown in the figure) is assumed to be external and connected via a wireless network to the *Multimedia Service*

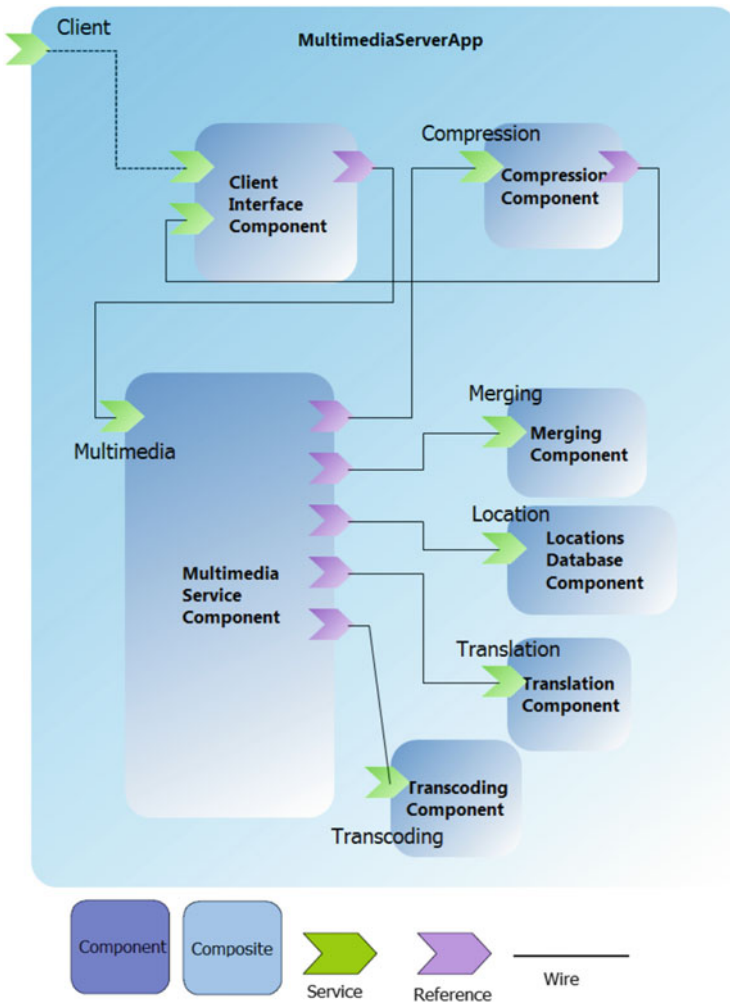


Fig. 4 SCA assembly of the Multimedia Service App

```

module ClientInterfaceComponent
...
rule r_ClientInterfaceComponent=
...
if nextRequest(self)="r_Requiring_news_in_textual_format" then
  seq
    r_receive[client(self),"r_Requiring_news_in_textual_format",requestParams(self)]
    if (isDef(requestParams(self))
      then
        seq
          r_sendreceive[multimedia(self),"requestNews",requestParams(self),responseParams(self)]
          r_replay[client(self),"r_Requiring_news_in_textual_format",responseParams(self)]
        endseq
      endseq
    endseq
  ...

```

Fig. 5 Behavior of the SCA-ASM component `ClientInterfaceComponent` for providing news in textual format

App. The *Multimedia Service App* is a composite application. The component *ClientInterface* processes information on the client’s behalf. It coordinates the service *MultimediaService* and also with the service *Compression* to adapt the news content to the wireless link. The service *MultimediaService* interacts with the services: *Transcoding* to adapt the video content for the smartphone format, *Translation* to adapt the text for the smartphone format and draw the geographical map, the service *Merging* to integrate the text with the video stream for the limited smartphone display, and *Locations Database* to collect information about the localization of cells and thus provide a map showing the user location.

As an example, Fig. 5 shows a simplified fragment of the service operation “Requiring news in textual format” of the core component *ClientInterface* (as provided by the exposed service interface *Client*) using SCA-ASM. It corresponds to the functionality (1) to provide the news ticker in textual format only. Note that, with respect to the BPEL-based description that contains the orchestration of the overall components interactions, from the point of view of the *Client* component’s behavior, only the delegation to the *Multimedia* component (by the invocation of the service operation “requestNews” of the *Multimedia* service through a `sendReceive` interaction) and the replay of the result to the client are visible.

5 Reliability Model for BPEL

This section presents a state-of-the-art [12, 44] reliability model for Web service composition expressed in BPEL. This reliability model contains an optimization model for the service selection based on the response time, cost, and availability of a service. Through the composition of elementary software services (or *abstract* services), a BPEL composition, modeling a Web service-based system, offers *K* services (or *external* services) to users. Each external service, or BPEL process, can be graphically represented as a tree structure, according to the following definition:

Definition 1 A BPEL process k that represents the external service k can be seen as a direct acyclic graph $DAG_k = (V_k, E_k)$, where the nodes V_k are the BPEL activities and the edges E_k represent the relationships among the BPEL activities.

Specifically, an internal node $i \in V_k$ represents a structured activity. These nodes have labels l in the set $\{seq., switch, while, flow\}$, where the symbol $seq.$ denotes (for brevity) the sequential execution. Similarly, a leaf node $i \in V_k$ is associated with the invocation (i.e., with a primitive *invoke* activity) of an elementary service s_i .

Figure 6 shows an example of BPEL tree for an external service provided by the interaction of four elementary software services. In particular, the figure represents the BPEL tree of the one of the functionalities provided by the smartphone application. The corresponding BPEL representation is listed in Fig. 2. Notice that the p_a labels denote the probability of executing activity a in a structured activity (e.g., in Fig. 6, p_{c1} and p_{c2} are the probabilities to execute the activities in the switch statement).

For each elementary service s_i , invoked in a BPEL process k , several *concrete* services may exist that match its description. We assume that the instances available for the service s_i are functionally compliant with it, i.e., each instance provides at least all the services provided by s_i and requires at most all the services required by s_i . Instances of the same service may differ for cost and reliability characteristics. We call J_i the set of instances for s_i , while s_{ij} represents the j th instance of J_i .

A *service broker* acts as an intermediary for the matching between the abstract elementary services and the concrete services—between service requestors and providers. Shortly, the broker defines the business process for the composite service and discovers and selects the best concrete services in order to minimize the costs and guarantee a given level of system reliability. As described in [12], the broker’s architecture consists of different modules, which interact with each other. Examples are the *Composition Manager* that is responsible for the service composition and

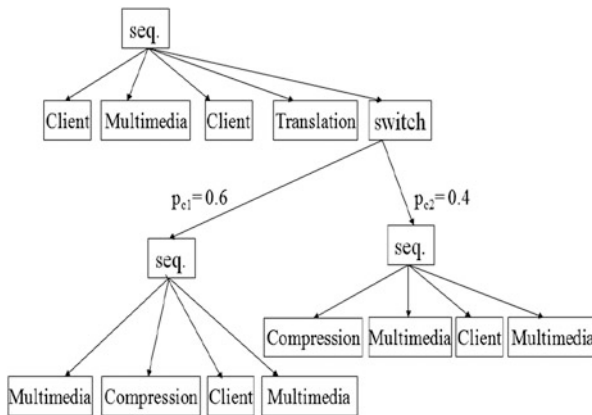


Fig. 6 BPEL tree of the external service “Requiring news in textual format”

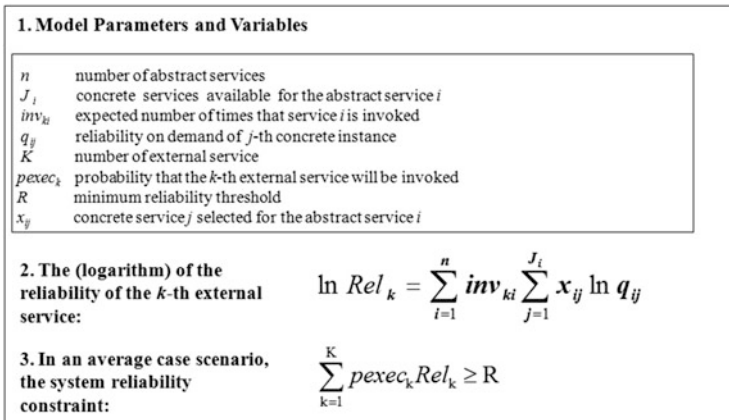


Fig. 7 BPEL-based reliability model

the concrete services’ discovery and the *BPEL Engine* module that provides the software platform to execute the business process described in BPEL. The broker performs a centralized orchestration, i.e., it represents the single coordinator node for the execution of the Web service specification. Readers interested in the broker’s details can refer to [12].

Figure 7 shows the reliability model for the Web service selection optimization problem. In particular, the figure shows how to predict the reliability of an external service k , starting from its BPEL activity tree.

Let $pexec_k$ be the probability that the k th system external service will be invoked. This information can be synthesized from the usage profile [32].⁴ The reliability of the k th external service⁵ can be computed by combining (1) the expected number of times inv_{ki} that the service s_i is invoked in the BPEL tree k and (2) the reliability q_{ij} of the concrete service s_{ij} selected in J_i for s_i .

6 Reliability Model for SCA-ASM

This section presents a reliability model for an SCA assembly involving SCA-ASM components. In particular, we assume that for each service exposed by the SCA assembly (composite component), there is an SCA-ASM component, a “core”

⁴The usage profile may be not (fully) available. In such cases, the domain knowledge and the information provided by the SOA could be used for estimating it, as suggested, for example, in [38].

⁵For the sake of model linearity, as in [44], when writing expressions, we consider the logarithm of the reliability rather than the reliability itself.

1. Model Parameters and Variables	
K	number of services exposed and provided by the assembly
$pexec_k$	probability that the k -th service will be invoked
C_k	set of components orchestrated by the core component M_k
n_c^i	probability that the component c is invoked (logarithm of the) reliability of the component c
$r_{M_k}^{-i}$	(logarithm of the) reliability of the component M_k , which reflects the nature of failures in an ASM.
2. The reliability of an assembly SCA:	$Rel = \sum_{k=1}^{ K } pexec_k \cdot Rel_{M_k}$
3. The reliability of a run run_k, of length n of M_k:	$Rel_{M_k} = \prod_{i=0}^{n-1} Rel_{M_k}^i$
4. The reliability of SCA-ASM core component M_k:	$Rel_{M_k}^i = e^{(\sum_{c=1}^{ C_k } n_c^i \cdot r_c^i) + r_{M_k}^{-i}}$

Fig. 8 Reliability model for SCA-ASM

component, that provides that service on behalf of the composite by interacting with and coordinating the other SCA subcomponents (possibly implemented with different programming languages). Moreover, we assume that such SCA-ASM components are single-agent ASMs; hence, we do not consider dynamic instantiation of subagents (by the use of the *spawn rule*).

As done in BPEL reliability model, the reliability model we present considers only crash failures and assumes that service invocation is synchronous.

Figure 8 reports, in a concise way, the SCA-ASM reliability model originally presented in [37]. The reliability formulas rely on the representation of the SCA-ASM core component's program⁶ as tree structure and on the concept of SCA-ASM usage profile.

We premise, therefore, the following definition of rule dependency tree (RDT) and of usage profile.

Let k be a service provided by the SCA-ASM core component of an SCA assembly, and let M_k be the SCA-ASM model of this component.

Definition 2 We call *RDT* of M_k the structure $RDT_k = (V_k, E_k)$ where the nodes V_k are the rule invocations in the program M_k and the edges E_k reflect the nesting relationship among these rule invocations according to the rule constructors for computation/coordination.

⁶We recall from [36] that an SCA-ASM component has a distinguished rule name of arity zero, taking by convention the same name as the component (e.g., rule r_A in Fig. 3 for component A). This rule is assigned as a program to the component's agent created during the initialization, and it is used as entry point for the component execution.

Note that basic computation rules and communication rules are associated with leaf nodes, while computation/coordination rule constructors are associated with internal nodes. Hence, for each non-root node $i \in V_k$, its parent node $f(i)$ is the rule constructor within which rule i occurs. Note that an RDT can be defined for any named rule of M_k , but we treat such trees as subtree of the “main RDT” by collapsing the nodes corresponding to their invocations (i.e., rule call nodes).

Definition 3 The SCA-ASM usage profile of the service k is the tuple

$$(pexec_k, (P(rule_1), P(rule_2), \dots, P(rule_m)))$$

where:

$pexec_k$ is the probability of execution for service k and can be given in input from the designer or can be derived from the observation of the SCA-ASM component behavior or from observations derived from systems offering the same type of services.

$P(rule_j)$ is the probability of executing $rule_j$, $1 \leq j \leq m$, supposing m is the number of occurrences of rule constructors in the SCA-ASM model for service k .⁷ These probabilities are derived from historical data deriving from the observation of the SCA-ASM component behavior.

On the tree structure RDT_k , we introduce a labeling function $l, E_k \rightarrow [0, 1]$ defined as follows: each edge $(f(i), i) \in E_k$ is labeled with $l(f(i), i) = p_i$ where p_i is the probability that the rule i is executed (e.g., the probabilities 0.7 and 0.3 in Fig. 9).

Such probability values can be derived by the usage profile of the service k .

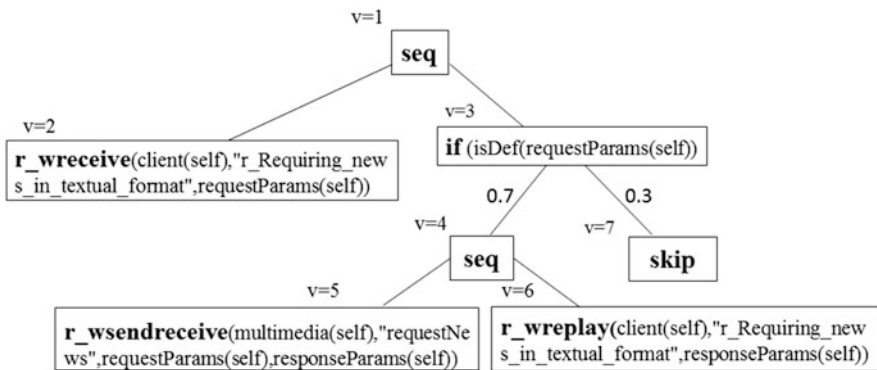


Fig. 9 RDT of the SCA-ASM component *ClientInterface* for providing news in textual format

⁷More precisely, m yields the number of rule constructors occurring in the *rules*, *services*, and *program* of the SCA-ASM component providing the service k .

Figure 6 shows the RDT corresponding to the fragment of the core SCA-ASM component *ClientInterface* reported in Fig. 5. Unlabeled edges have a probability value 1.

6.1 Reliability Model Formulation

Let S be the SCA assembly of $|C|$ components and K be the number of services exposed and provided by S . For each exposed service k of S , one of the $|C|$ components provides the service k .

Let $pexec_k$ be the probability that the k th service will be invoked. It must hold $pexec_k \geq 0$ for all $k = 1 \dots K$ and $\sum_{k=1}^K pexec_k = 1$. This information can be synthesized from the operational profile [32].

The reliability of the overall assembly SCA yields

$$\text{Rel} = \sum_{k=1}^{|K|} pexec_k \cdot \text{Rel}_{M_k}, \quad (1)$$

where Rel_{M_k} is the reliability of the provided service k .

Under failure independence assumption, it can be modeled as follows:

$$\text{Rel}_{M_k} = \prod_{i=0}^{n-1} \text{Rel}_{M_k}^i, \quad (2)$$

where n is the length of the execution run_k of the machine M_k and $\text{Rel}_{M_k}^i$ is the reliability of the move mov_i (a single computation step from state s_{i-1} to state s_i) of the machine M_k .

Considering the (logarithm of the) reliability is additive [14], the reliability of the move mov_i of M_k can be computed as

$$\text{Rel}_{M_k}^i = e^{(\sum_{c \in C_k} n_c^i r_c^i) + \bar{r}_{M_k}^i} \quad (3)$$

where (1) $C_k \subset C$ is the set of components orchestrated by the core SCA-ASM component for providing the service k ; (2) n_c^i is the expected number of times that the component $c \in C_k$ is invoked; (3) r_c^i is the logarithm⁸ of the reliability of the SCA component c , namely, the probability that the components complete its task when invoked; and (4) $\bar{r}_{M_k}^i$ is the logarithm of the reliability of the core SCA-ASM component, which reflects the nature of failures in an ASM.

The parameter n_c^i can be easily estimated by parsing the paths of the RDT_k of the core component, from the root to the leaves containing primitives `wsend-`

⁸For the sake of model linearity, as in [44], we consider the *logarithm* of the reliability rather than the reliability itself.

receive/receive to/from c , and multiplying the p_i labels of the edges along the paths.

To compute the reliability of an SCA-ASM component, and thus the probability $\bar{r}_{M_k}^i$, in [37] we show a precise way by considering failures specific to the nature of ASMs: occurrence of *inconsistent updates*⁹ and violation of *invariants*.¹⁰ According to these crash failures, the reliability of an SCA-ASM component c is given by

$$\text{Rel}_c^i = \text{Rel}_{IU}^i \cdot \text{Rel}_{IM}^i \cdot \text{Rel}_{Ie}^i, \quad (4)$$

where Rel_{IU}^i is the SCA-ASM reliability for *inconsistent update failures*, whereas Rel_{IM}^i and Rel_{Ie}^i are the SCA-ASM machine and the environment reliabilities, respectively, for *invariant failures*.

In case the component c is not an SCA-ASM component, its reliability r_c must be given. Suggestions to estimate r_c can be found in [18]. A rough upper bound $1/N_{nf}$ can be estimated upon observing that the component has been invoked for N_{nf} number of times with no failures.

7 Comparing BPEL-Based and SCA-ASM Reliability Models

In order to compare the reliability models of BPEL and SCA-ASM, we first need to map concepts of one notation into concepts of the other, and we then make an evaluation of the two reliability models by performing some experiments on the selected case study.

7.1 Mapping Concepts

In Table 3, we compare the features of an SCA assembly and a composite Web service.

In Table 4, we compare the features of the RDT of the k th service provided by the SCA assembly and BPEL activity tree of the k th service provided by the composite Web service, while Table 5 shows the features of the reliability model of the k th service provided by the SCA assembly and the reliability model of the k th service provided by the BPEL model.

⁹Let us recall (see Definition 2.4.5 in [6]) that *consistency of updates* guarantees that an ASM location is never simultaneously updated to different values.

¹⁰Recall that an invariant expresses a constraint one wants to assume for some functions of the ASM signature. Such constraints are stated as first-order formulas that have to hold in every state of the ASM.

Table 3 Comparing SCA assembly and BPEL composition features

SCA Assembly		BPEL	
C	Set of SCA components	S	Set of web services
K	Number of services provided by the assembly	K	Number of services provided by the service composite
p_{exec_k}	Probability that the service k will be invoked	p_{exec_k}	Probability that the service k will be invoked

Table 4 Comparing SCA-ASM rule tree and BPEL activity tree features

SCA-ASM rule tree $RDT_k = (V_k, E_k)$		BPEL activity tree $DAG_k = (V_k, E_k)$	
V_k	rules in the SCA-ASM model	V_k	activities in the BPEL code
<i>internal node</i>	rule constructor	<i>internal node</i>	structured activity
<i>leaf node</i>	basic rule for computation or communication (correspond to a SCA comp. invocation)	<i>leaf node</i>	invoke activity (correspond to an elementary service invocation)
w_i	prob. of executing rule i in a rule constructor	p_a	prob. of executing activity a in a structured activity

Table 5 Comparing SCA assembly reliability model and BPEL-based reliability model features

SCA assembly rel. model		BPEL-based rel. model	
n_i^c	prob. that the component c is invoked	inv_{ki}	number of time that the service i is invoked
<i>internal node</i>	rule constructor	<i>internal node</i>	structured activity
<i>leaf node</i>	basic rule for computation or communication (correspond to a SCA comp. invocation)	<i>leaf node</i>	invoke activity (correspond to an elementary service invocation)

We recall that an SCA assembly represents a service-oriented architecture. Therefore, the usage profile of the external service k —the one related to the run_k (see Sect. 6)—provided by the SCA-ASM assembly, represents the usage profile of the service k provided by the composite Web service.

7.2 Experimental Evaluation

In this section, we show the numerical results obtained by comparing the SCA-ASM reliability model and the BPEL-based model referring to the example presented in Sect. 3. We first describe the generation of the model parameters and then we describe different experimental results.

7.2.1 Model Parameters

Following a standard approach for parametric evaluation, we first define a set of so-called nominal parameters, which constitute the starting point of the experimentation, and then we generate random instances by perturbing the nominal values. The average, maximum, and minimum values of the different obtained results are then considered.

As shown in the software architecture of the running example in Fig. 4, we have associated the IDs with the services as follows: s_1 to Client, s_2 to Multimedia Service, s_3 to Locations Database, s_4 to Transcoding, s_5 to Translation, s_6 to Merging, and s_7 to Compression.

Table 6 shows the starting parameter values of the available instances for the abstract services. Three different external services using them are considered. The third column of Table 6 lists the set of alternatives for each existing service. For each alternative, the reliability r_{ij} is given in the third column; the expected number of times inv_{1i} that the service i is invoked within the first external service is given in the fourth column; the expected number of times inv_{2i} that the service i is invoked within the second external service is given in the fifth column; the number of times inv_{3i} that the service i is invoked within the third external service is given in the fifth column.

Table 6 Parameters of the available instances for the existing services

Service ID	Service altern.	Reliability r_{ij}	Num. of inv. first scen. inv_{1i}	Num. of inv. second scen. inv_{2i}	Num. of inv. third scen. inv_{3i}
s_1	s_{11}	0.9993	3	2	2
	s_{12}	0.9994			
	s_{13}	0.99999			
s_2	s_{21}	0.996	3	2.084	3
	s_{22}	0.9995			
s_3	s_{31}	0.99985			1.2
s_4	s_{41}	0.999		1	
	s_{42}	0.99985			
	s_{43}	0.99999			
s_5	s_{51}	0.9993	1	1	1
	s_{52}	0.9998			
	s_{53}	0.99998			
s_6	s_{61}	0.94		1.084	
	s_{62}	0.9997			
	s_{63}	0.9999			
s_7	s_{71}	0.99	1	1.084	1
	s_{72}	0.992			
	s_{73}	0.999999			

Starting from the services’ nominal values, we have generated 186 different system configurations (here also called *perturbed configurations*) by randomly changing the parameters. Two of these configurations differ for concrete services’ reliabilities, which we have slightly decreased/increased (e.g., within 10 % of the nominal values).

In order to generate the perturbed configurations, we have generated five concrete service bases. For each abstract service, the number of concrete services spans from 13 to 65. Two service bases differ for the number—spanning from 13 to 65—and the parameters of the perturbed configurations (i.e., one perturbed configuration corresponds to seven concrete services generated for the seven abstract services). For the sake of result robustness, the service bases’ parameters have been varied.

7.2.2 Numerical Results

Case 1 In Fig. 10, we report the results obtained applying the SCA assembly model without orchestrators and the BPEL-based reliability model to the perturbed configurations. We have fixed the probabilities (i.e., p_{exec_k}) that the first, second, and third external services will be invoked with probabilities 0.3, 0.3, and 0.4, respectively. Each bar—corresponding to one service base—contains the higher, lower, and the average values obtained for the system reliability. The results are the same for both models.

Case 2 In Fig. 11 we illustrate the results obtained considering the SCA assembly, taking into account also the service orchestrator’s reliability.

For each perturbed configuration, we have applied our SCA assembly reliability model for a set of reliability values for the three external services’ orchestrators. Each orchestrator’s reliability spans from 0.95 to 0.999 by steps of 0.005. Each group of 17 bars—corresponding to one service base—refers to the execution model results over the base’s perturbed configurations. In particular, each bar corresponds

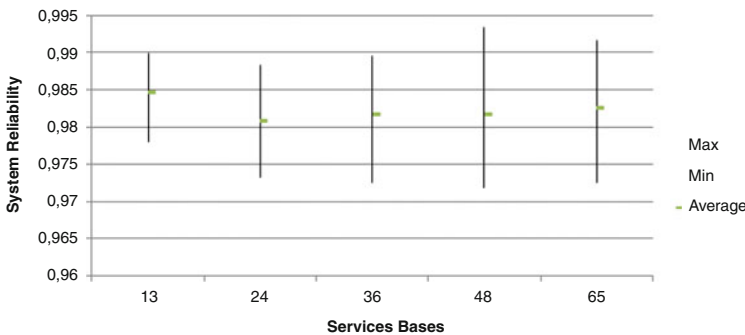


Fig. 10 System reliability obtained with the BPEL reliability model and with the SCA assembly without considering the orchestrator’s reliabilities

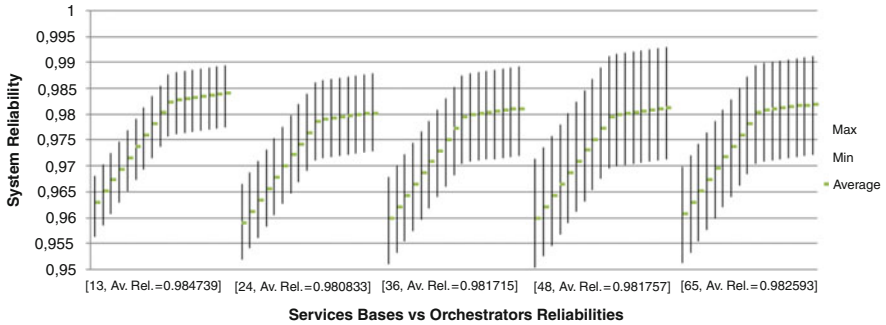


Fig. 11 System reliability obtained for the SCA assembly considering the orchestrator’s reliabilities

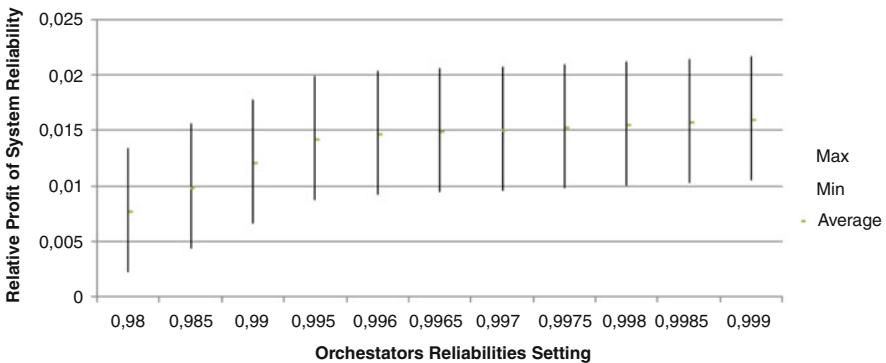


Fig. 12 Comparing the relative profit of different orchestrator’s reliabilities setting

to one fixed reliability value for the reliability of the three service orchestrators and contains the higher, lower, and the average system reliability. The lower (higher or average) system reliability increases while increasing the orchestrator’s reliabilities. For example, with a service base of 13 perturbed configurations, if the reliability of the three orchestrators is equal to 0.95, the average system reliability decreases from 0.984739 (average value estimated without considering the orchestrator’s reliabilities in case 1) to 0.963045, whereas if the reliability of all orchestrators increases to 0.995, then the system reliability is about 0.982597.

These results show the relevance of the orchestrator’s reliability in a reliability model. These assertions can be better evaluated by considering Fig. 12, where it is shown the percentage gain in system reliability obtained with the orchestrator’s reliabilities setting from 0.98 to 0.999 by steps 0.005. We have compared the system reliability obtained with these reliability values and the orchestrator’s reliabilities setting from 0.95 to 0.975 by steps 0.005. Specifically, for a perturbed configuration s , we have estimated the profit as follows: $(RelSys_v(s) - RelSys_{v'}(s))/RelSys_v(s)$, where $RelSys_v(s)$ and $RelSys_{v'}(s)$ represent the system reliability returned by the

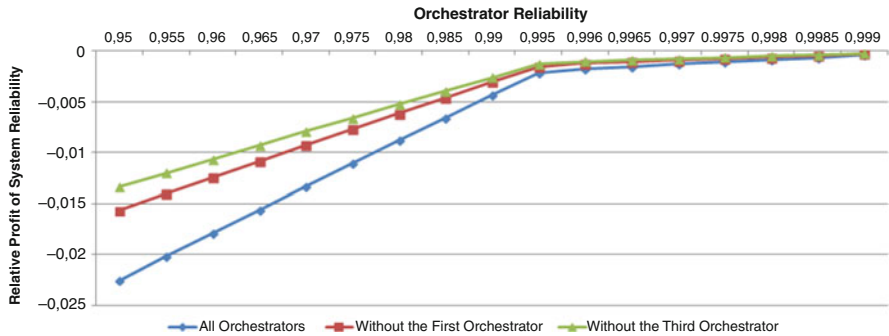


Fig. 13 Comparing the relative profit of the system reliability

SCA assembly reliability model by considering the orchestrator’s reliabilities fixed to v and v' , respectively. The figure shows the results for the service base with 65 perturbed configurations.

As expected, the lower (higher or average) gain in system reliability increases while increasing the orchestrator’s reliability value. For example, with the reliability of the orchestrators fixed to 0.98, the average profit is about 0.0078, whereas if the reliability of the orchestrators increases to 0.997, then the average profit increases, and it is about 0.01518.

Case 3 In order to compare the efficacy of considering the orchestrator’s reliabilities, we have analyzed in Fig. 13 the gain in reliability—in terms of a more precise (and less optimistic) estimation—obtained in three different experiments.

The first experiment—corresponding to the curve with diamonds—refers to the execution of SCA assembly by considering the reliability of all orchestrators (i.e., the results obtained in case 2). The second experiment—corresponding to the curve with rectangles—refers to the execution of SCA assembly reliability model by considering the reliability of the first service’s orchestrator equal to 1. Finally, the third experiment—corresponding to the curve with triangles—refers to the execution of SCA assembly reliability model where the reliability of the third service’s orchestrator is equal to 1. The figure shows the results for the service base with 48 perturbed configurations. The x -axis represents the variation of the orchestrator’s reliability. Specifically, each point—corresponding to one reliability orchestrator’s value—contains the average relative profit obtained for the system reliability.

A relevant observation is that for the corresponding values of curves in the three experiments, the average profit is higher in the first experiment than in the second and third ones. This is because in the first experiment, we consider the reliabilities of all service orchestrators, whereas in the second and third experiment, we fix to 1 the reliability of one of the service orchestrators. For example, for the first experiment, with the reliabilities of the orchestrators equal to 0.97, the (modulus)

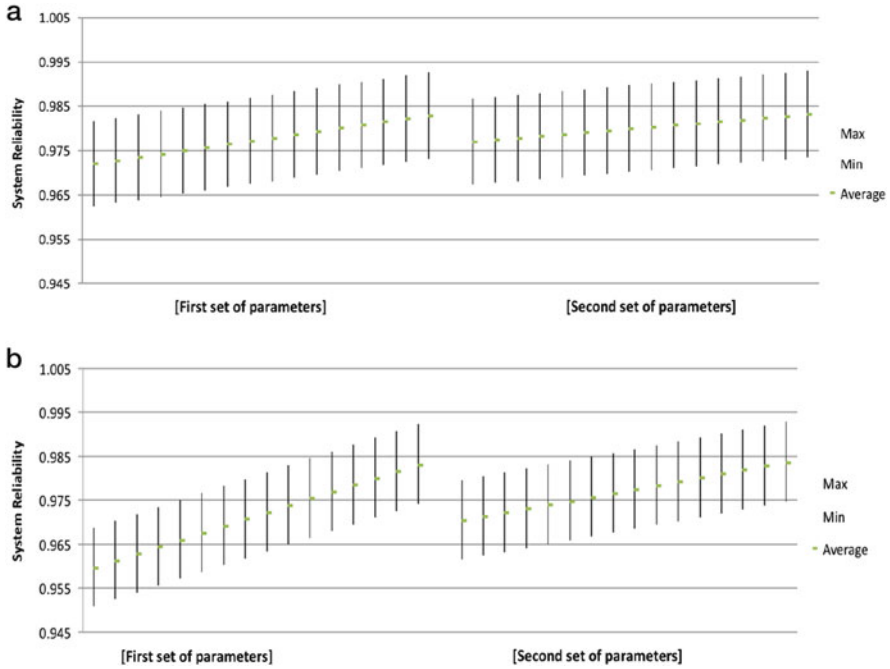


Fig. 14 System reliability obtained by embedding the reliability of an SCA-ASM component. (a) Considering the service s_3 as an SCA-ASM component. (b) Considering the service s_7 as an SCA-ASM component

average profit is about 0.01332, whereas for the second and third experiments, the (modulus) average profit is about 0.00928 and 0.00795, respectively.

On the other hand, for corresponding values of curves in the second and third experiments, the average profit is higher in the second experiment than in the third one. For example, for the second experiment, with the orchestrator’s reliabilities equal to 0.98, the (modulus) average profit is about 0.0061, whereas for the third experiment, the (modulus) average profit is about 0.0052. This is due to the fact that in the second experiment, we consider the reliability of the orchestrator of the first external service equal to 1 (with the probability to be invoked fixed to 0.3), whereas in the third experiment, we consider the reliability of the orchestrator of the third external service equal to 1 (with the probability to be invoked fixed to 0.4).

Case 4 Figure 14 shows the results we obtain when in the SCA assembly we explicitly model the reliability of an SCA-ASM component, considering the service base of 65 perturbed configurations. Specifically, Fig. 14a, b illustrates the results obtained by explicitly using our SCA-ASM reliability model for s_3 and s_7 , respectively. In both cases, we have observed the system reliability while varying the reliability of s_3 and s_7 (assuming that it is due to inconsistent update failures) and considering two different set of parameters.

Each bar—corresponding to a single reliability value—contains the higher, lower, and the average value obtained for the system reliability. The observed different behavior of the reliability model in the two cases is mainly due to the fact that services s_3 and s_7 have a different probability to be invoked in the system. In fact, service s_3 is only used in the third external service, whereas service s_7 is used in all external services (see Table 6).

These results highlight how the reliability model that exploits the specific features of an SCA-ASM component leads to a more precise (and less optimistic) reliability estimation.

8 Related Work

A wide range of approaches have been proposed for the reliability and availability analysis of Web services in both analytical models (e.g., [23, 29, 39]) and empirical studies (e.g., [25]). The analytical models exploited different kinds of Markov processes to define availability/reliability models for a composite Web service. The empirical analyses considered both workloads and the reliability of Web servers, proposing to distinguish between intersession and intra-session Web characteristics. More recently, some papers tackled the problem of composing a service-oriented system from publicly available Web services (e.g., [45]), taking into account different types of Web service failures. Several approaches use notations based on BPEL (e.g., [8, 43], and [12] discussed in Sect. 5).

Concerning quality estimation of Web services based on the standard SCA, the work in [20] presents a reliability computation for the SCA component model. It proposes a dynamic behavior model for specifying the component interface behavior by a notion of *port* and *port activities*. It defines failure behaviors of ports through the *nonhomogeneous Poisson process* (NHPP). Thus, the overall system reliability is computed on the reliability of port expressions.

Using a general component model, the work in [7] proposes a reliability prediction method based on the Palladio Component Model (PCM), which offers a UML-like modeling notation. In particular, a tool is defined to automatically transform PCM models into Markov chains.

Formal methods, such as Petri net, automata, and process algebra, are also used by existing approaches for quality evaluation (see, e.g., the work in [19] for performance modeling notations). Ad hoc models (such as UML), used to describe the static and dynamic aspects of a software system, are typically transformed in formal quality models (such as queueing networks). Thus, a major problem of these approaches may reside in the distance between notations for modeling the system and notations for modeling qualities.

As far as the reliability is concerned, a quite extensive list of approaches can be found in literature (e.g., see survey [28]). Most of these approaches (e.g., [7, 17], and [16]) use notations based on UML sequence and deployment diagrams annotated with reliability properties, such as failure probabilities. Tools can transform such high-level models into analysis models (such as Markov models for state-based approaches), which then can be evaluated. As an example, the KLAPER suite [16] is a modeling framework (language, methodology, tools) for the predictive modeling and analysis of performance and reliability of component-based systems. It uses model transformations to automatically generate analysis models (queueing networks) out of an annotated UML design model.

As remarked in [27], the formal models typically focus only on the formal modeling of the service functionality and behavior. In [27], the formal specification of services with context-dependent contracts and their compositions is provided. Nonfunctional aspects are also taken into account, and the model checking technique is exploited to verify service properties w.r.t. the composition specification. The verification of functional and nonfunctional aspects based on a formal method is also performed in [1], where process algebraic techniques are used for architecture-level functional and performance analysis.

9 Conclusions and Future Work

In this chapter, we have presented two reliability prediction methods for service-oriented applications: one based on the BPEL language [3] and the other based on the service-oriented component model SCA-ASM [31, 35–37].

We have performed a comparison between these two different approaches by conducting a wide experimental analysis. The obtained results show that when considering the same modeling abstraction level, the two reliability models are equivalent. However, since the SCA-ASM reliability model allows a more detailed modeling and computation of the reliability of a single Web component, the reliability estimation of the overall Web service system with SCA-ASM provides more accurate results than the ones obtained with the BPEL-based model. The comparison suggested that further benefit can be obtained by combining the use of the two models. Indeed, while the BPEL-based reliability model can be used for the evaluation of the whole process, the SCA-ASM reliability model can be used to evaluate the reliability of the single services, thus providing more accurate results than the one obtained with the current BPEL-based reliability models.

To further study the scalability and the representativeness of these models, we plan to apply these approaches to other examples and to compare their predicted reliability values with existing data of real-life experiments.

References

1. Aldini, A., Bernardo, M., Corradini, F.: A Process Algebraic Approach to Software Architecture Design. Springer, London (2010)
2. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient QoS-aware service composition. In: WWW, pp. 881–890 (2009)
3. Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Golan, Y., Kartha, N., König, D., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0, OASIS Standard Specification, 11 April 2007. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
4. Apache Tuscany: <http://tuscany.apache.org/> (2011)
5. Arcaini, P., Gargantini, A., Riccobene, E., Scandurra, P.: A model-driven process for engineering a toolset for a formal method. *J. Softw. Pract. Exp.* **41**(2), 155–166 (2011)
6. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, New York (2003)
7. Brosch, F., Koziolok, H., Buhnova, B., Reussner, R.: Architecture-based reliability prediction with the palladio component model. *IEEE Trans. Softw. Eng.* **38**(6), 1319–1339 (2012)
8. Bruneo, D., Distefano, S., Longo, F., Scarpa, M.: QoS assessment of WS-BPEL processes through non-Markovian stochastic Petri nets. In: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp. 1–12 (2010)
9. Brugali, D., Gherardi, L., Riccobene, E., Scandurra, P.: Coordinated execution of heterogeneous service-oriented components by abstract state machines. In: Arbab, F., Ölveczky, P.C. (eds.) FACS. Lecture Notes in Computer Science, vol. 7253, pp. 331–349. Springer, Berlin (2011)
10. Business Process Model and Notation: <http://www.bpmn.org/> (2012)
11. Calinescu, R., Ghezzi, C., Kwiatkowska, M.Z., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. *Commun. ACM* **55**(9), 69–77 (2012)
12. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F.: Flow-based service selection for web service composition supporting multiple QoS classes. In: ICWS, pp. 743–750. IEEE Computer Society, Salt Lake City (2007)
13. Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Lo Presti, F., Mirandola, R.: MOSES: a framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Trans. Softw. Eng.* **38**(5), 1138–1159 (2012)
14. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. *J. Web Semant.* **1**(3), 281–308 (2004)
15. Chandran, S.K., Dimov, A., Punnekkat, S.: Modeling uncertainties in the estimation of software reliability. In: 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI), pp. 227–236. IEEE Computer Society, Singapore (2010)
16. Ciancone, A., Filieri, A., Luigi Drago, M., Mirandola, R., Grassi, V.: KlaperSuite: an integrated model-driven environment for reliability and performance analysis of component-based systems. In: TOOLS (49). Lecture Notes in Computer Science, vol. 6705, pp. 99–114. Springer, Berlin (2011)
17. Cortellessa, V., Potena, P.: Path-based error propagation analysis in composition of software services. In: Software Composition. Lecture Notes in Computer Science, vol. 4829, pp. 97–112. Springer, Berlin (2007)
18. Cortellessa, V., Marinelli, F., Potena, P.: Automated selection of software components based on cost/reliability tradeoff. In: EWSA. Lecture Notes in Computer Science, vol. 4344, pp. 66–81. Springer, Berlin (2006)
19. Cortellessa, V., Di Marco, A., Inverardi, P.: Model-Based Software Performance Analysis. Springer, Berlin (2011)
20. Ding, Z., Jiang, M.: Port based reliability computing for service composition. In: Proceedings of the 2009 IEEE International Conference on Services Computing, SCC '09, pp. 403–410 (2009)

21. Filieri, A., Ghezzi, C., Grassi, V., Mirandola, R.: Reliability analysis of component-based systems with multiple failure modes. In: CBSE. Lecture Notes in Computer Science, vol. 6092, pp. 1–20. Springer, Berlin (2010)
22. Gargantini, A., Riccobene, E., Scandurra, P.: A metamodel-based language and a simulation engine for abstract state machines. *J. Universal Comput. Sci.* **14**(12), 1949–1983 (2008)
23. Gokhale, S.S., Lu, J.: Performance and availability analysis of an E-commerce site. In: 30th Annual International Computer Software and Applications Conference, 2006. COMPSAC '06, vol. 1, pp. 495–502 (2006)
24. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. *Perform. Eval.* **45**(2–3), 179–204 (2001)
25. Goseva-Popstojanova, K., Deep Singh, A., Mazimdar, S., Li, F.: Empirical characterization of session-based workload and reliability for web servers. *Empir. Softw. Eng.* **11**(1), 71–117 (2006)
26. Grassi, V.: Architecture-based reliability prediction for service-oriented computing. In: WADS. Lecture Notes in Computer Science, vol. 3549, pp. 279–299. Springer, Berlin (2004)
27. Ibrahim, N., Mohammad, M., Alagar, V.S.: An architecture for managing and delivering trustworthy context-dependent services. In: IEEE SCC, pp. 737–738 (2011)
28. Immonen, A., Niemelä, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Softw. Syst. Model.* **7**(1), 49–65 (2008)
29. Janevski, N., Goseva-Popstojanova, K.: Session reliability of web systems under heavy-tailed workloads: an approach based on design and analysis of experiments. *IEEE Trans. Softw. Eng.* **99**(PrePrints), 1 (2013)
30. Krka, I., Edwards, G., Cheung, L., Golubchik, L., Medvidovic, N.: A comprehensive exploration of challenges in architecture-based reliability estimation. In: Architecting Dependable Systems VI. Lecture Notes in Computer Science, vol. 5835, pp. 202–227 (2009)
31. Mirandola, R., Potena, P., Riccobene, E., Scandurra, P.: A reliability model for service component architectures. *J. Syst. Softw.* **89**, 109–127 (2014)
32. Musa, J.D.: Operational profiles in software-reliability engineering. *IEEE Softw.* **10**(2), 14–32 (1993)
33. OASIS/OSOA: Service component architecture (SCA). www.oasis-open.org/sca (2011)
34. Potena, P.: Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff. *J. Syst. Softw.* **86**(3), 624–648 (2013)
35. Riccobene, E., Scandurra, P.: A formal framework for service modeling and prototyping. *Form. Asp. Comput.* **26**(6), 1077–1113 (2014)
36. Riccobene, E., Scandurra, P., Albani, F.: A modeling and executable language for designing and prototyping service-oriented applications. In: EUROMICRO-SEAA, pp. 4–11. IEEE, Oulu (2011)
37. Riccobene, E., Potena, P., Scandurra, P.: Reliability prediction for service component architectures with the SCA-ASM component model. In: Cortellessa, V., Muccini, H., Demirörs, O. (eds.) EUROMICRO-SEAA, pp. 125–132. IEEE Computer Society, İzmir (2012)
38. Roshandel, R., Medvidovic, N., Golubchik, L.: A Bayesian model for predicting reliability of software systems at the architectural level. In: QoS. Lecture Notes in Computer Science, vol. 4880, pp. 108–126. Springer, Berlin (2007)
39. Sato, N., Trivedi, K.S.: Accurate and efficient stochastic reliability analysis of composite services using their compact markov reward model representations. In: IEEE International Conference on Services Computing, 2007. SCC 2007, pp. 114–121 (2007)
40. Smith, C.U., Williams, L.G.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley, Redwood City (2002)
41. Standard Glossary of Software Engineering Terminology: STD-729-1991 ANSI/IEEE (1991)
42. The ASMETA Toolset Website: <http://asmeta.sf.net/> (2011)
43. Xia, Y., Liu, Y., Liu, J., Zhu, Q.: Modeling and performance evaluation of BPEL processes: a stochastic-petri-net-based approach. *IEEE Trans. Syst. Man Cybern. A Syst. Hum.* **42**(2), 503–510 (2012)

44. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.* **30**, 311–327 (2004)
45. Zheng, Z., Lyu, M.R.: Collaborative reliability prediction of service-oriented systems. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE '10*, pp. 35–44. ACM, New York (2010)

What Constitutes a Service on the Web?

Towards a Theory of Services

Klaus-Dieter Schewe and Qing Wang

Abstract There are many approaches to modelling and development of service-oriented systems, but there is still no convincing answer to what a (software) service is. In this chapter, we discuss the various attempts to develop a theory of services, identify aspects that have already been addressed and extract gaps. This leads us to propose the BDCM² framework capturing *behaviour*, *description*, *contracting*, *monitoring* and *mediation*. For the behavioural model, we refer to the two-layer model of Abstract State Services (AS²s) capturing functional aspects of data-intensive Web services. The model of service mediators permits building complex applications, in which parts are realised by services. Furthermore, we highlight the decisive role of service ontologies for supporting the location of services as well as the capture of contractual aspects by means of service-level agreements (SLAs). Finally, we conclude that a contract is only as good as the means to monitor the agreements. While part of the research has already reached a promising level of maturity, some aspects are still in an infant state.

1 Introduction

It is now commonly accepted that the main role of the World Wide Web is to provide a pool of services rather than documents. Such a service can in fact be anything: a simple function, a data warehouse or a fully functional information system. However, this brings with it the fundamental question what actually is a service in general or in a more restrictive sense, what is a software service on

K.-D. Schewe

Software Competence Center Hagenberg, Softwarepark 21, 4232 Hagenberg, Austria

Christian Doppler Laboratory for Client-Centric Cloud Computing, Johannes Kepler University Linz, Softwarepark 21, 4232 Hagenberg, Austria

e-mail: kd.schewe@scch.at; kd.schewe@edcc.faw.jku.at

Q. Wang (✉)

Research School of Computer Science, The Australian National University, Canberra ACT 0200, Australia

e-mail: qing.wang@anu.edu.au

© Springer International Publishing Switzerland 2015

B. Thalheim et al. (eds.), *Correct Software in Web Applications and Web Services*,

Texts & Monographs in Symbolic Computation,

DOI 10.1007/978-3-319-17112-8_8

the Web? There are many approaches trying to address this fundamental question [1, 3, 7–9, 16, 17, 22, 24–27, 30, 33, 36, 42, 45]. According to Bergholtz et al. [9], these approaches can be roughly classified into those taking a business-centric view, i.e. the focus is on business services, and those taking a software-centric view, i.e. the focus is on software services. Examples of the former class are among others the service science framework by Ferrario et al. [16], the so-called unified theory of services (UTS) by Sampson et al. [36], the approach to semantic Web services by Preist [33] and the Advancing Open Standards for the Information Society (OASIS) framework by Alves et al. [3]. Examples of the latter class are among others the various attempts to address service-oriented architectures (SOAs) [4, 13, 31], service-oriented computing (SOC) [32], Web services [2, 6, 41, 43] and the behavioural model of Abstract State Services [25, 26]. Both lists can be extended by many other examples (see, for instance, the literature review in [26] or in [9]). Some of the approaches have indeed a hybrid character, i.e. they approach at the same time the business and the software-technical aspects.

In this article, we want to continue our research towards the fundamental question “what constitutes a service”; summarise the rationale underlying our work so far, which led to the model of Abstract State Services (AS²s) [26], the model of service mediators based on AS²s [28, 39, 40] and the service ontology model [27]; and discuss which further properties characterise a (software) service (on the Web). Originally (as postulated in our first article on the AS² model [25]) our objective was to lay foundations of a theory of service-oriented systems. In particular, we claimed that the following fundamental questions should be answered:

- How must a general model for services look like capturing the basic idea and all facets of possible instantiations, and how can we specify such services?
- How can we search for services that are available on the Web?
- How do we extract from such services the components that are useful for the intended application, and how do we recombine them?
- How can we optimise service selection using functional and non-functional (aka “quality of service”) criteria?

1.1 Functional Behaviour

Regarding the first question, we deviated significantly from approaches taken by the research on SOA, SOC or Web services. We adopted the viewpoint that a “behavioural theory” for services—more precisely software services that are made available (publicly or for a restricted audience) via the Web—should be approached. This notion on “behavioural theory” has been coined only recently by Blass and Gurevich to cover the research on the abstract state machine (ASM) thesis [10, 19] and follow-on work [38].

According to the SOC research roadmap service foundations, service composition, service management and monitoring and service-oriented engineering have

to be addressed [32]. For research on SOC and Web services, many researchers have adopted the Web services description language (WSDL) proposed by W3C [12] or the OASIS Web services standard [3] as the starting point. That is, what is considered to be a service is already linguistically fixed, and on these grounds, the listed problems are addressed. The most investigated research directions concern Web service integration [6] and service personalisation [18]. Similarly, for SOA, a common view is that Web services [2] shall be identified, managed by means of a hierarchical representation and composed and orchestrated to create complex application systems [31]. Again, the WSDL [12] takes the role of describing the parameters that are needed to use a Web service; service publishing is addressed by a Universal Description, Discovery and Integration (UDDI) registry [43], which addresses universal description, discovery and integration; service location is supported by service ontologies such as Web Service Modelling Ontology (WSMO) [15] based on the Web Services Modelling Framework (WSMF) [14] or any other ontology dedicated to Web services; and service orchestration exploits languages such as the Business Process Execution Language (BPEL).

A “behavioural theory” does not start from a language that somehow represents a reasonable class of the objects that are to be characterised, i.e. in our case the services. Instead, a language-independent clarification of the desired notion is given by means of a set of intuitive postulates. For sequential algorithms, this was done by Gurevich leading to the sequential ASM thesis [19]; for unbounded parallel algorithms, this was done analogously by Blass and Gurevich leading to the (still debated) parallel ASM thesis [10]. For non-deterministic database transformations, we contributed a language-independent characterisation exploiting meta-finite structures [38]. In the second step, an abstract machine model—notably (sequential, parallel, database) abstract state machines (ASMs)—is defined, for which it is then proven in the third step that the machine model captures exactly the objects stipulated by the postulates.

This led to the notion of Abstract State Service (AS²) [26]. That is, we defined postulates for the (functional) behaviour of a (software) service and proved that a particular machine model, in this case a federation of variants of ASMs (i.e. a distributed ASM), captures exactly the postulates. Then of course, any equivalent language (by means of expressiveness) could be used to implement the theory of AS²s. We will briefly discuss this behavioural theory of services in Sect. 2.

However, the AS² model only covers functional aspects of services. Roughly speaking, an AS² provides a process that can be used by someone else knowing only what the process of implementing the service is supposed to do. The user does neither own the service nor is he/she able to manipulate it. While this covers some aspects of services—to be precise, it covers *behaviour*—it is far from being a complete model of services. In the sense of the discussion of fundamental work on services by Bergholtz et al., AS²s might be considered as some form of abstraction of what constitutes a service [8, 9], though it provides a much more fundamental theoretical model than OASIS, semantic Web services or the approaches to service science developed so far [3, 24, 33].

1.2 Service Ontologies

The second question above concerns the location of services, for which a description of the available services is needed. The key idea is that given a coarse description of the service needed, it should be possible to search for such a service. This is exactly what ontologies are meant to capture, a description of the (semantics of) services. Therefore, we adopt the common idea of a service ontology, which is already omnipresent in the area of the semantic Web, also in our own work in [27, 28]. This is usually grounded in some more or less expressive description logic [5], e.g. the Web Ontology Language (OWL) [44]. In general, description logics are not the most expressive logical languages, as only unary and binary predicates are used and junctors as well as quantifiers are restricted—this is discussed in detail in the description logics handbook and the many references listed in it [5]. However, the reason for preferring description logics over more expressive logical languages is that in such a logic, the implication between unary concepts is decidable, which permits automatic classification, i.e. each new class of services or each individual service will be placed correctly into a hierarchy.

While it is not possible to precisely describe a service (e.g. using the AS² model) in a way that it can be automatically matched to a service request—as services represent complex software systems in general, this is a well-known undecidable property—it is commonly accepted that a service ontology should capture three aspects (e.g. see [14–16, 27, 30] and the references in there):

- *Functional* description: This should cover the specification of input and output types as well as pre- and postconditions telling in technical terms what the service will do.
- *Categorical* description: This should capture interrelated keywords telling what the service operation does by using common terminology of the application area.
- *Quality of service (QoS)* description: This should capture non-functional properties such as availability, response time, cost, etc.

A functional description alone would be insufficient. For instance, a flight booking service may functionally be almost identical to a train booking system. Therefore, an additional categorical description is indispensable. The terminology of the application domain defines an ontology in the widest sense, i.e. we have to provide definitions of “concepts” and relationships between them, such that each offered service becomes an instantiation of one or several concepts in the terminology. As remarked in our previous work, setting up a categorical ontology for services in a particular application domain is already a tedious and time-consuming endeavour. Furthermore, such an ontology only makes sense if a larger community agrees on it. However, in many technical application areas, there is quite reluctance to make details of application knowledge publicly (or at least semipublicly) available. On the other hand, in areas such as biology or chemistry, commonly accepted ontologies exist already for a long time.

According to our previous work, the QoS description is not needed for service discovery and merely useful to select among alternatives, but neither functional nor categorical description can be dispensed with [27]. While the first part of this statement is still true, we would argue now that the role of non-functional properties should be seen in a wider context. The non-functional properties cover also the description of how a service is meant to be used; what are the obligations and rights of the participating agents, at least of the service provider and the service user; how conflicts are to be handled; etc. The fact that these non-functional aspects should be considered as being intrinsically part of a service leads to additional aspects of our fundamental question concerning what services are.

Thus, service ontologies capture a second aspect of a general service model, the availability of a detailed *description* of the service. We will address the functional and categorical aspects of a service ontology in Sect. 3. In this context, we will also discuss quality of service as part of a wider discussion of contractual aspects, obligations and rights. For the latter, we will look also into the discussion of properties that are distinctive for service in comparison to related research [9, 16, 36, 45].

1.3 Service Mediation

For the third question above, we first observed that in the AS² model, the specification of how a service is to be used was left implicit [28]. Therefore, in order to make the workflow within a service explicit, we formalised the notion of *plot* of a service, which actually captures the possible sequencing of service operations. For this we first exploited Kleene algebras with tests (KATs) [28], which are known to be the most expressive formalism to capture propositional process specifications. Then we permitted more details of the AS² to be revealed in a generalised plot by using ASMs with abstract submachines [40]. So adding plots to the AS² model is a little extension of the behavioural model, which in addition impacts on the functional description in the service ontology. The notion of plot is a term adopted from the movie business that has already been used for a long time in the context of Web information systems (WISs) [37].

Service mediation addresses collaboration of services, which to our point of view is another necessary aspect of services. While the functional behaviour of a service is entirely in the hands of a service provider and the service description addresses how a service is offered by the provider to the potential users, service mediation covers how users can make use of a service. For instance, it is commonly known that online sales services are often used as information repositories without the slightest intention to buy something. In other words, while the functionality of a service is defined by the provider and conditions of use are subject of the QoS part of the service ontology, it is exclusively up to the user to exploit the service for his/her own purposes.

Therefore, we consider service mediation as an important aspect of a theory of services. For this we introduced service mediators, which exploit plots with open slots for services to specify intended service-based applications on a high level of abstraction [28, 39]. The idea is to specify service-oriented applications that involve yet unknown component services. On these grounds, matching criteria for services are formally defined that are to fill the slots. A problem in finding such matching criteria is the fact that it should be possible to skip component operations of services and change their order. This enhances the work on service composition, which is already a well-explored area in service computing with respect to services that are understood functionally. In the AS² model, this corresponds to the service operations rather than the services as a whole. More precisely, what actually needs to be composed are “runs” of services that are determined by the plot including conditions, under which particular service operations can be removed or their order can be changed. This led to rather complicated matching conditions between services and slots in mediators [28, 39, 40].

We will discuss the aspect of service *mediation* in Sect. 4. We will only refer to the mediator model as such and abstract from any details how an instantiated mediator could be realised across multiple service repositories. For this, the research on client-cloud interaction on grounds of ambient ASMs provides deeper insights [11].

1.4 Service Contracts

The aspects of behaviour, description and mediation alone do not yet capture everything that would characterise services. It is not surprising that a lot of researches concerning the characterisation of “service science” or “conceptual models of services” do not stress the functional and usage aspects at all. Different from other approaches, the AS² model including the service plots provides a behavioural theory capturing functional aspects of services, but as many engineering approaches to service specification, composition, orchestration, etc., would qualify as implementations of AS²s, the formal clarity achieved will not invalidate these approaches. Similarly, the necessity to have functional and categorical parts in service ontologies is commonly accepted, so the clarification, in which formal basis is needed—this is also an open problem in the context of the AS² model—will smoothen the edges of the theory, but it will not make extreme distinctions.

This is different regarding the non-functional aspects of services. Zeithaml et al. emphasise general properties such as intangibility, inseparability, heterogeneity and perishability of services [45], which in the community have been controversially debated [8, 16, 36] and rejected as being neither necessary nor sufficient. Indeed, intangibility refers to the fact that a service is owned by its provider, while a service user can exploit the service for his/her own purpose, but there is never a transfer of ownership nor does the user even know how the service is realised. For instance, in the often-used example of a snow removal service, it is up to the provider to

decide whether shovels, brooms or snow ploughs are used, which the service client is not interested in at all, as long as the agreed result is guaranteed. Intangibility is de facto captured by the behavioural model; in particular, in the AS² model, a hidden internal layer is separated from the visible layer exposed to the user, which includes the associated plot. Similarly, perishability is no property of services at all. Every software system is prone to perishability, if the hardware and system software it is grounded in disappear. In the contrary, it should be part of the usage agreement between a service provider and a user that services do not perish within the agreed period of service usage. That is, availability agreements are part of the service as well as the consequences, if the agreement is violated. Inseparability has to be treated also with care. Of course, from the point of view of the provider, the service is offered as a whole, and there is an agreement about this with each service user. However, as already discussed in connection with mediation, it is up to the user to exploit the services in his/her own way and for his/her own purposes, regardless of what the provider intended. In this sense, a user may well cut out of a service the parts that are really needed, even though for the provider it still appears that the service was used as a whole. Finally, heterogeneity is not a characteristic at all, as it could only refer to the collection of all services, in which case it is a triviality. If it were to refer to a single service, there is no reason for the claim that heterogeneity of the components involved should always be the case.

Sampson et al. in the UTS and also Bergholtz et al. emphasise the usage and exchange of resources within a context and the shift of the management of resources to the provider as important characteristics of services [8, 9, 36], while Ferrario et al. stress legal aspects associated with provision and usage of services [16]. We consider this a limited perspective, which stresses a rather indecisive and in general debatable view that the management of resources and the rules governing it are of central importance.

Therefore, let us first take a closer look into the service model promoted by Bergholtz et al. [8, 9], before developing our key idea regarding *contracting* as a decisive feature of a theory of services. As Bergholtz et al. do not claim universal validity for their conceptual service model, we cannot conclude that their approach is misleading, but we would like to look at the properties of the model from a different angle, which will (hopefully) lead us to a sharpened view regarding the desired theory of service. The model is grounded in an ontology capturing resources, events and agents, which is coupled with a classification of rights. The original resource-event-agent (REA) ontology was developed by McCarthy [29] and then further extended by Geerts and McCarthy [17] and Hruby [21]. The classification of rights goes back to the early work by Hohfeld [20] but is also tangent to the work by Ferrario et al. concerning legal aspects of services [16].

As the name already indicates, the used REA ontology stresses resources, events and agents as the main building blocks. Resources are roughly classified as being internal, shared, consumable, etc. Similarly, events are classified into conversion, production, consumption, exchange, etc. The agents refer to the individuals, groups, machines, etc., that perform the events on the resources. Regarding the resources, it is at least debatable whether resources are involved each time in a service. For

instance, when seeking legal advice, the asked lawyer may qualify as an agent involved in the picture, but hardly as a resource, and the access to a book reference may not be needed at all. What is much more characteristic than the use of resources is the fact that a workflow is issued when a service is used. That is, instead of talking about events, it would be much more convincing to talk about the workflow resulting from a process—though this may be considered a matter of terminology—and the agents involved in this workflow. The workflow as such is nothing more than what we called the (functional) behaviour, and the agents involved refer to the implementation of this behaviour, which is not decisive. What is decisive is that there is a service provider offering the service and a user of the service. If any third party is involved and this is important for the agreements between the provider and the user, then this becomes part of the regulations governing the service use. That is, in our opinion, the specific REA ontology mainly addresses functional and categorical aspects of the service ontology. Classifying types of services regarding production, consumption, conversion, exchange, etc., may be helpful, but it does not remove the burden that even for a limited application area the set-up of an ontology and the achievement of a common agreement about it remain a hard task. Any general concepts for such an ontology may be helpful, but the decisive characteristic is that a description of functional and categorical aspects of services is part of the service model.

The second aspect concerns the classification of rights following Hohfeld. The most interesting feature here is the fact that rights governing the use of services are considered part of the service model, whereas the classification of the rights remains on the surface. In general, rights could be expressed by deontic action rules similar to the specification of rights and obligations as part of the conceptual model of WISs [37]. That is, a deontic action logic would be required, in which such rights (and also obligations) could be expressed in connection with the involved actors. In doing so, the QoS description in the service ontology will have to capture these deontic rules governing the provision and usage of services.

However, we feel that this would still draw an incomplete picture regarding non-functional aspects, as there are more rules than those capturing rights and obligations. In general, there should be a whole list of SLAs, which altogether define a contract between the provider of a service and a service user. Rady has defined fragments of an ontology capturing SLAs [34] and implemented a tool extracting contracts from such an SLA ontology [35]. For the time being, the SLA ontology, which in our opinion should be part of the service ontology, only addresses availability and performance aspects, though more than 20 additional types of SLAs have already been discussed in the literature (see also [35] for a brief overview). SLAs concerning availability, performance, etc., but also security and privacy regulations have nothing to do with rights. For instance, availability on one hand concerns conditions of usage, e.g. if the service is only available on workdays within a specified time period, and on the other hand a commitment and obligation by the provider. Security and privacy may be also handled as commitments, but it should preferably also include the means with which security is supposed to be achieved, in which case the SLA includes a statement about the functional

behaviour of the service or its environment. In summary, for each service, there must exist a service contract capturing all SLAs, and the SLAs may be expressed by obligations and rights in a deontic action logic, refer to functional aspects or simply cover factual data. The decisive feature, however, is that a model of services must comprise the *contracting* aspect. We will discuss contracting in Sect. 3, though our own research in this direction has not yet progressed very far.

1.5 SLA Monitoring

Finally, we would like to emphasise that a contract that cannot be validated is rather useless, both technically and legally. Therefore, for every service it should be possible to check not only its behaviour but also whether the SLAs are fulfilled. That is, there must exist a monitoring software for this purpose—this could again be a service, but not necessarily. Monitoring as such is an established field, in particular with respect to performance. However, a systematic connection of monitoring with SLAs is still missing. The work by Lampesberger and Rady regarding monitoring of SLAs (in the context of cloud computing) [23] is a promising step in this direction. For our discussion here, we only conclude that *monitoring* is a decisive aspect in a theory of services, which has been almost completely neglected so far. As this part is still very immature, we dispense with a discussion of SLA monitoring in this chapter.

1.6 Summary

In our analysis of relevant related research addressing the fundamental question of what constitutes a service, we highlighted five decisive aspects:

- *Behaviour*: There must exist a general behavioural theory of services.
- *Description*: There must exist a description of a service that allows it to be discovered and used.
- *Contracting*: There must exist a contract between the service provider and user covering all relevant SLAs for the service.
- *Mediation*: A user must use a service in the contracted way but can build service mediations to realise service-centric applications satisfying his/her purposes.
- *Monitoring*: It must be possible to monitor the execution of a service in order to validate its behaviour and contracted SLAs.

Therefore, we refer to our approach to a general theory of (software) services (on the Web) as the *BDCM*² framework. In the following sections, we will discuss the framework in more detail. Then we conclude with a brief summary and outlook in Sect. 5.

2 Abstract State Services: A Behavioural Theory of Services

The AS^2 model provides a behavioural theory, so it comes first with a set of postulates.

2.1 AS^2 Postulates

Each (software) service will provide some form of workflow that will access data resources. Therefore, the AS^2 model refers to an underlying database, using this term in a very general sense. Traditional database architecture distinguishes at least three layers: a conceptual layer describing the database schema in an abstract way, a physical layer implementing the schema and an external layer made out of views. The external layer exports the data that can then be used by users or programs. For our purposes here, we can neglect the physical layer, but in order to capture services, we complete this architecture by adding operations on both the conceptual and the external layer; the former is handled as database transactions, whereas the latter provides the means with which users can interact with a database.

In order to abstract from this architecture to obtain a model of abstract services, we first formulate postulates for the database layer. Following the general approach of ASMs [19], we may consider each database computation as a sequence of abstract states, each of which represents the database (instance) at a certain point in time plus maybe additional data that is necessary for the computation, e.g. transaction tables, log files, etc. In order to capture the semantics of transactions, we distinguish between a wide-step transition relation and small-step transition relations. A transition in the former one marks the atomic execution of a transaction, so the wide-step transition relation defines infinite sequences of transactions. Without loss of generality, we can assume a serial execution, while of course interleaving is used for the implementation, as long as this is equivalent to the serial execution, i.e. serialisability is guaranteed. Then each transaction itself is a database transformation and as such corresponds to a finite sequence of states resulting from a small-step transition relation, which should then be subject to the postulates for database transformations [38]. We will explain these postulates later in this section.

Definition 1 (Database Postulate) A *database system* (DBS) consists of the following:

- A set \mathcal{S} of states, together with a subset $\mathcal{I} \subseteq \mathcal{S}$ of initial states
- A wide-step transition relation $\tau \subseteq \mathcal{S} \times \mathcal{S}$
- A set \mathcal{T} of transactions, each of which is associated with a small-step transition relation $\tau_t \subseteq \mathcal{S} \times \mathcal{S}$ ($t \in \mathcal{T}$) satisfying the postulates of a database transformation over \mathcal{S}

With this definition, we do not yet specify what states are. We do, however, already require that states of a database system are states of database transformations. Later, when we discuss the postulates for database transformations, we will further elaborate the notion of state and database transformation.

For now, note that differently from the sequential time postulate in Gurevich's work, we permit non-determinism both in the wide-step transition relation and in the small-step transition relations. For the first one, this is due to the fact that transactions may be started anytime, and the database system will schedule them in a serialisable way, thereby defining a (serial) run. The non-determinism in the small-step transition relations is far more limited, as it is mainly meant to capture the creation of values such as identifiers as a highly expressive means in query and update languages. This form of non-determinism is common in database transformations.

Definition 2 A *run* of a database system (DBS) is an infinite sequence S_0, S_1, \dots of states $S_i \in \mathcal{S}$ starting with an initial state $S_0 \in \mathcal{S}$ such that for all $i \in \mathbb{N}$ $(S_i, S_{i+1}) \in \tau$ holds, and there is a transaction $t_i \in \mathcal{T}$ with a finite run $S_i = S_i^0, \dots, S_i^k = S_{i+1}$ such that $(S_i^j, S_i^{j+1}) \in \tau_{t_i}$ holds for all $j = 0, \dots, k - 1$.

Example 1 Let us consider a flight booking system. At its core, it may use a database storing data about flights and bookings. For simplicity, assume that we use a relational database, so we may have a relation FLIGHT with attributes flight_no, departure_date, departure_time, origin and destination for the available flights; a relation SEAT with attributes flight_no, departure_date, class and number for the available seats per class in a flight; and BOOKING with attributes booking_ref, flight_no, departure_date, class and customer_id for the already-made bookings. Let us ignore everything else such as customer data, status of bookings, etc.

Then a state of the DBS would contain an instance of the relational database schema, and a booking transaction would change the state by adding further tuples to the booking relation, provided the number of seats booked for each class of each flight does not exceed the number of available seats. The booking transaction itself proceeds stepwise, and each step also changes the database, i.e. the state.

Furthermore, a booking may be issued by a customer after receiving an answer to a query, e.g. asking for flight itineraries from a specified origin airport to a destination airport within a specified timeframe. The answer to such a query would be a set of itineraries, and each itinerary would be specified by a set of flight tuples stored in the database. Thus, the state, in which the booking transaction is started, should also contain the set of itineraries, which is a view on top of the relational database.

The preceding example is of course very simplified, but it illustrates the definition of a database system. Note that if views are considered as part of states of a DBS, then transactions also affect them.

Views in general are expressed by queries, i.e. read-only database transformations. Therefore, we can assume that a view on a database state $S_i \in \mathcal{S}$ is given by a finite run S_0^v, \dots, S_ℓ^v of some database transformation v with $S_0^v = S_i$ and

$S_i \subseteq S_\ell^v$. Traditionally, we would consider $S_\ell^v - S_i$ as the view. Here we assume that we can write a state of a database system as a set. For instance, if we deal with a relational database system, then each relation is a set of tuples, which can be written as first-order atoms, and the whole database is the union of these sets of atoms. We will later explain that it is also possible in general to view a state as a set.

We can use this to extend a database system by views. For this, let each state $S \in \mathcal{S}$ to be composed as a union $S_d \cup V_1 \cup \dots \cup V_k$ such that each $S_d \cup V_j$ is a view on S_d . As a consequence, each wide-step state transition becomes a parallel composition of a transaction and an operation that switches views on and off. This leads to the definition of an Abstract State Service (AS²).

Definition 3 (Extended View Postulate) An *Abstract State Service* (AS²) consists of a database system (DBS), in which each state $S \in \mathcal{S}$ is a finite composition $S_d \cup V_1 \cup \dots \cup V_k$ and a finite set \mathcal{V} of (extended) views. Each view $v \in \mathcal{V}$ is associated with a database transformation such that for each state $S \in \mathcal{S}$, there are views $v_1, \dots, v_k \in \mathcal{V}$ with finite runs $S_0^j, \dots, S_{n_j}^j$ of v_j ($j = 1, \dots, k$), starting with $S_0^j = S_d$ and terminating with $S_{n_j}^j = S_d \cup V_j$. Each view $v \in \mathcal{V}$ is further associated with a finite set \mathcal{O}_v of (service) operations o_1, \dots, o_n such that for each $i \in \{1, \dots, n\}$ and each $S \in \mathcal{S}$, there is a unique state $S' \in \mathcal{S}$ with $(S, S') \in \tau$. Furthermore, if $S = S_d \cup V_1 \cup \dots \cup V_k$ with V_i defined by v_i and o is an operation associated with v_k , then $S' = S'_d \cup V'_1 \cup \dots \cup V'_m$ with $m \geq k - 1$, and V'_i for $1 \leq i \leq k - 1$ is still defined by v_i .

In a nutshell, in an AS², we have view-extended database states, and each service operation associated with a view induces a transaction on the database and may change or delete the view it is associated with and even activate other views. We therefore talk of views that are *open* and those that are *closed*.

Example 2 The booking operation in Example 1 is a service operation that is associated with a view that produces a list of itineraries for given search criteria such as origin and destination, preferred class, departure timeframe, etc. The induced transaction on the DBS updates the BOOKING relation. Initial states for this database transformation can be any consistent database plus any set of open views. The successor state (for the wide-step transition relation τ) would contain the updated database and the same set of views except the one containing the list of itineraries, which would be replaced by a view that simply contains a confirmation message for the selected and booked itinerary.

2.2 Postulates for Database Transformations

The definition of database systems and by that also the definition of AS² refer to postulates for database transformations that have been elaborated in [38]. We will briefly describe these postulates here, though a full motivation will not be possible due to space limitations. In total, there will be five postulates: the

sequential time postulate, the *abstract state postulate*, the *background postulate*, the *bounded exploration postulate* and the *bounded non-determinism postulate*. An object satisfying these postulates will be a data transformation. Together with the *database postulate* in Definition 1 and the *extended view postulate* in Definition 3, we obtain the complete definition of AS²s by means of postulates.

Definition 4 (Sequential Time Postulate) A database transformation t is associated with a non-empty set of states \mathcal{S}_t together with non-empty subsets \mathcal{I}_t and \mathcal{F}_t of the initial and final states, respectively, and a one-step transition relation τ_t over \mathcal{S}_t , i.e. $\tau_t \subseteq \mathcal{S}_t \times \mathcal{S}_t$.

Analogously to Definition 2, a *run* of a database transformation t is a finite sequence S_0, \dots, S_f of states with $S_0 \in \mathcal{I}_t$, $S_f \in \mathcal{F}_t$, $S_i \notin \mathcal{F}_t$ for $0 < i < f$ and $(S_i, S_{i+1}) \in \tau_t$ for all $i = 0, \dots, f - 1$.

The abstract state postulate is an adaptation of the corresponding postulate for ASMs [19], according to which states are first-order structures, i.e. sets of functions. These functions are interpretations of function symbols given by some signature.

Definition 5 A *signature* Σ is a set of function symbols, each associated with a fixed arity. A *structure* S over Σ consists of a set B , called the *base set* of the structure together with interpretations of all function symbols in Σ , i.e. if $f \in \Sigma$ has arity k , then it will be interpreted by a function $f_S : B^k \rightarrow B$. An isomorphism σ from structure S to structure S' is defined by a bijection $\sigma : B_S \rightarrow B_{S'}$ between the base sets that extends to functions by $\sigma(f_S(b_1, \dots, b_k)) = f_{S'}(\sigma(b_1), \dots, \sigma(b_k))$.

Taking structures as states reflects common practice in mathematics, where almost all theories are based on first-order structures. Variables are special cases of function symbols of arity 0, and constants are the same, but unchangeable. We will later in the background postulate formulate the minimum requirements for the base set such as containing truth values, a value representing undefinedness and more.

Definition 6 (Abstract State Postulate) All states $S \in \mathcal{S}_t$ of a database transformation t are structures over the same signature Σ_t , and whenever $(S, S') \in \tau_t$ holds, the states S and S' have the same base set. The sets \mathcal{I}_t , \mathcal{I}_t and \mathcal{F}_t are closed under isomorphisms, and for $(S_1, S'_1) \in \tau_t$, each isomorphism σ from S_1 to S_2 is also an isomorphism from S'_1 to $S'_2 = \sigma(S'_1)$ with $(S_2, S'_2) \in \tau_t$.

Furthermore, the signature Σ_t is composed as a disjoint union out of a database signature Σ_{db} , an algorithmic signature Σ_a and a finite set of unary bridge function symbols, i.e. $\Sigma_t = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$. The base set of a state is $B = B_{db} \cup B_a$ with interpretation of function symbols in Σ_{db} and Σ_a over B_{db} and B_a , respectively. The interpretation of bridge function symbols defines a function from B_{db} to B_a . With respect to such states, the restriction to Σ_{db} is a finite structure.

Example 3 In the booking Example 1, we have to deal with finite relations FLIGHT, SEAT and BOOKING, so for the database part, a finite structure would be sufficient. However, in the service operations including the view-defining queries, we may need to permit arithmetic operations such as counting, adding prices, determining

the time difference between arrival and departure, etc., for which we would require the whole set of natural or real numbers. Thus, these infinite sets of numbers have to become part of the set B_a , and in each view that exploits values from these sets, we use surrogate identifiers instead, which can be drawn from the finite set B_{db} , and a bridge function assigning the actual values to the surrogate identifiers.

Another example arises, if we use finite Extensible Markup Language (XML) trees. In this case, each node in the tree would be represented by an identifier, and the tree structure would be expressed by order relations for SUCCESSOR and SIBLING. Thus, B_{db} would have to contain the set of hereditarily finite trees over a finite set \mathcal{O} of node identifiers. For the actual values associated with the tree nodes, we would provide a bridge function.

The major purpose for the explicit constructors in database transformations is the need to capture the constructs of data models. For instance, in complex-value and object-oriented databases, we may require the presence of constructors for records, finite sets, lists, multisets, disjoint unions, arrays, maps, etc. Starting from a set of base domains such as *Integer*, *Date*, *Bool*, etc., we can apply these constructors and nest them arbitrarily to define complex-value domains. In tree-based databases such as XML databases, we may even require a colimit constructor leading to hereditarily finite trees, i.e. the domain of all finite trees with nodes in a given base domain such that all subtrees are also trees in the same domain.

Definition 7 Let \mathcal{D} be a set of base domains and V_K a background signature; then a *background class* \mathcal{K} with V_K over \mathcal{D} is constituted by:

- The universe $\mathcal{U} = \bigcup_{D \in \mathcal{D}} D$ of elements, where \mathcal{D} is the smallest set with $\mathcal{D} \subseteq \mathcal{Q}$ satisfying the following properties for each constructor symbol $\lrcorner \lrcorner \in V_K$:
 - If $\lrcorner \lrcorner \in V_K$ has unfixed arity, then $\lrcorner D \lrcorner \in \mathcal{D}$ for all $D \in \mathcal{D}$, and $\lrcorner a_1, \dots, a_m \lrcorner \in \lrcorner D \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \dots, a_m \in D$.
 - If $\lrcorner \lrcorner \in V_K$ has unfixed arity, then $A_{\lrcorner \lrcorner} \in \mathcal{D}$ with $A_{\lrcorner \lrcorner} = \bigcup_{\lrcorner D \lrcorner \in \mathcal{D}} \lrcorner D \lrcorner$.
 - If $\lrcorner \lrcorner \in V_K$ has bounded arity n , then $\lrcorner D_1, \dots, D_m \lrcorner \in \mathcal{D}$ for all $m \leq n$ and $D_i \in \mathcal{D}$ ($1 \leq i \leq m$), and $\lrcorner a_1, \dots, a_m \lrcorner \in \lrcorner D_1, \dots, D_m \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \dots, a_m \in D$.
 - If $\lrcorner \lrcorner \in V_K$ has fixed arity n , then $\lrcorner D_1, \dots, D_n \lrcorner \in \mathcal{D}$ for all $D_i \in \mathcal{D}$ ($1 \leq i \leq n$), and $\lrcorner a_1, \dots, a_n \lrcorner \in \lrcorner D_1, \dots, D_n \lrcorner$ for all $a_1, \dots, a_n \in D$.
- An interpretation of function symbols in V_K over \mathcal{U}

Example 4 The view in Example 1 is to present a set of itineraries, in which each element is a list of flights. In order to model the necessary domain elements, we used constructors $[\cdot]$ and $\{\cdot\}$ for finite lists and finite sets, respectively, both with unfixed arity. Furthermore, we may use a constructor (flight_no, date, departure, origin, destination, class) of fixed arity six.

If *Flight_number*, *Date*, *Time*, *Airport* and *Character* denote base domains, then (flight_no:*Flight_number*, date:*Date*, departure:*Time*, origin:*Airport*, destination:*Airport*, class:*Character*) defines a complex domain for flights. Let this be called *Flight*. Then $\{[Flight]\}$ defines the domain for the set of itineraries.

That is, given the base set of a structure \mathcal{S} , we can add the required Booleans and \perp , partition it into base domains \mathcal{D} , apply the construction in Definition 7 to obtain a much larger base set and interpret function symbols with respect to this enlarged base set.

Definition 8 (Background Postulate) Each state of a database transformation t must contain:

- An infinite set of reserve values
- Truth values and their connectives, the equality predicate and the undefinedness value \perp
- A background class \mathcal{K} defined by a background signature V_K that contains at least a binary tuple constructor (\cdot) , a multiset constructor $\langle \cdot \rangle$ and function symbols for operations on pairs such as pairing and projection and on multisets such as empty multiset $\langle \rangle$, singleton $\langle x \rangle$ and multiset union \uplus .

For database transformations, computations are intrinsically parallel, even though implementations may be sequential, but the parallelism is restricted in the sense that all branches execute de facto the same computation. We will capture this by means of location operators, which generalise aggregation functions and cumulative updates.

Definition 9 Let $\mathcal{M}(D)$ be the set of all non-empty multisets over a domain D , then a *location operator* ρ over $\mathcal{M}(D)$ consist of a unary function $\alpha : D \rightarrow D$, a commutative and associative binary operation \odot over D and a unary function $\beta : D \rightarrow D$, which define $\rho(m) = \beta(\alpha(b_1) \odot \dots \odot \alpha(b_n))$ for $m = \langle b_1, \dots, b_n \rangle \in \mathcal{M}(D)$.

Example 5 A typical location operator is *count* counting the number of elements in a multiset. In this case, α assigns 1 to each element of D , \odot is addition, and β is the identity on D .

If α assigns to b the set $\{b\}$ and if b satisfies a formula φ and \emptyset otherwise, \odot is set union and β is again the identity, then the location operator defined by α , \odot and β assigns to a multiset $m \in \mathcal{M}(D)$ the set of elements in m satisfying φ .

The definitions of updates, update sets and update multisets are the same as for ASMs.

Definition 10 Let t be a database transformation and S be a state of t . A pair $(f, (a_1, \dots, a_n))$ consisting of an n -ary function symbol f and arguments a_1, \dots, a_n in the base set of S for its interpretation f_S in a state is called a *location*, usually written as $f(a_1, \dots, a_n)$. An *update* of t is a pair (ℓ, v) , where ℓ is a location $f(a_1, \dots, a_n)$ and v is another element in the base set of S . An *update set* is a set of updates; an *update multiset* is a multiset of updates.

Using a location function that assigns a location operator or \perp to each location, an update multiset can be reduced to an update set. It is further possible to construct for each $(S, S') \in \tau_t$ a minimal update set $\Delta(t, S, S')$ such that applying this update set to the state S will produce the state S' . Then $\Delta(t, S)$ denotes the set of all such update sets for t in state S , i.e. $\Delta(t, S) = \{\Delta(t, S, S') \mid (S, S') \in \tau_t\}$. The problem of partial updates is then subsumed by the problem of providing consistent update sets, in which there cannot be pairs (l, v_1) and (l, v_2) with $v_1 \neq v_2$ —details are discussed in [38].

The bounded exploration postulate in the sequential ASM thesis in [19] uses a finite set of ground terms as bounded exploration witness in the sense that whenever states S_1 and S_2 coincide over this set of ground terms, the update set produced by the sequential algorithm is the same in these states. The intuition behind the postulate is that only the part of a state that is given by means of the witness will actually be explored by the algorithm.

The fact that only finitely many locations can be explored remains the same for database transformations. However, permitting parallel accessibility within the database part of a state forces us to slightly change our view on the bounded exploration witness. For this, we need access terms, which in a sense cover associative access to databases.

Definition 11 An *access term* is either a ground term α or a pair (β, α) of terms, the variables x_1, \dots, x_n in which must be database variables, referring to the arguments of some dynamic function symbol $f \in \Sigma_{ab} \cup \{f_1, \dots, f_\ell\}$. The interpretation of (β, α) in a state S is the set of locations:

$$\{f(a_1, \dots, a_n) \mid \text{val}_{S,\zeta}(\beta) = \text{val}_{S,\zeta}(\alpha) \text{ with } \zeta = \{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}\}.$$

Structures S_1 and S_2 *coincide* over a set T of access terms if the interpretation of each $\alpha \in T$ and each $(\beta, \alpha) \in T$ over S_1 and S_2 is equal.

Instead of writing (β, α) for an access term, we should in fact write (f, β, α) , but for simplicity, we drop the function symbol f and assume it is implicitly given.

Due to our request that the database part of a state is always finite, there will be a maximum number m of elements that are accessible in parallel. Furthermore, there is always a number n such that n variables are sufficient to describe the updates of a database transformation, and n can be taken to be minimal. Then for each state S , the upper boundary of exploration is $\mathcal{O}(m^n)$, where m depends on S . Taking these together, we obtain our fourth postulate.

Definition 12 (Bounded Exploration Postulate) For a database transformation t , there exists a fixed, finite set T of access terms of t such that $\Delta(t, S_1) = \Delta(t, S_2)$ holds whenever the states S_1 and S_2 coincide over T .

The last postulate addresses the question of how non-determinism is permitted in a database transformation. To handle this, we need to further clarify the relationship between access terms and states. As defined in the abstract state postulate, every state of a database transformation is a meta-finite structure consisting of two parts:

the database part and algorithmic part, which are linked via a fixed, finite number of bridge functions. To restrict non-determinism in a database transformation t , we consider that *ground access terms* of t can access only the algorithmic part of a state, while *non-ground access terms* of t can access both the database and algorithmic parts of a state. Furthermore, variables in non-ground access terms are limited to range merely over the database part.

Given a meta-finite structure with the signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$ and the base set B , i.e. $B = B_{db} \cup B_a$, we now formally define access terms.

Definition 13 A *ground access term* is defined by the following rules:

- $\alpha \in B_a$ is a ground access term.
- $f(\alpha_1, \dots, \alpha_n)$ for n -ary function symbol $f \in \Sigma_a$ and ground access terms $\alpha_1, \dots, \alpha_n$ is a ground access term.

A *non-ground access term* is a pair (β, α) of terms in which at least one of them is a non-ground term inductively defined by applying function symbols from Σ over variables in accordance with the definition of a meta-finite structure.

We define equivalent substructures in the following sense.

Definition 14 Given two structures S' and S of the same signature Σ , a structure S' is a *substructure* of the structure S (notation: $S' \preceq S$) if the following holds:

- The base set B' of S' is a subset of the base set B of S , i.e. $B' \subseteq B$.
- For each function symbol f of arity n in the signature Σ , the restriction of $\text{val}_S(f(x_1, \dots, x_n))$ to B' results in $\text{val}_{S'}(f(x_1, \dots, x_n))$.

Substructures $S_1, S_2 \preceq S$ are *equivalent* (notation: $S_1 \equiv S_2$) if there exists an automorphism $\sigma \in \text{Aut}(S)$ with $\sigma(S_1) = S_2$. The *equivalence class* of a substructure S' in the structure S is the subset of all substructures of S which are equivalent to S' .

Now we formalise the bounded non-determinism postulate to capture these ideas by properly defining the presence of non-ground access terms. In doing so, we put a severe restriction on the non-determinism in the transition relation τ_t .

Definition 15 (Bounded Non-determinism Postulate) For a database transformation t , if there are states S_1, S_2 and $S_3 \in \mathcal{S}_t$ with $(S_1, S_2) \in \tau_t$, $(S_1, S_3) \in \tau_t$ and $S_2 \neq S_3$, then there exists a non-ground access term of the form (β, α) in the bounded exploration witness of t .

2.3 A Language for Abstract State Services

Let us now sketch an abstract language for the specification of AS^2 s—details were presented in [26]. We can specify the database layer by:

- A background class specifying additional base types, each associated with a base domain, constructor symbols and function symbols associated with these constructors
- A signature comprising function symbols for the database and algorithmic parts of states and for the bridge functions
- A set of initial states for the database system
- A set of transactions, each of which will be defined by a DB-ASM rule
- A set of auxiliary DB-ASM rules

On top of such specification of a database system, we define the view layer by a set of extended views. Each view is defined by:

- A signature defined similarly to the signature for the underlying database system
- A defining query that is defined by another DB-ASM rule possibly using auxiliary rules
- A set of operations that are specified similar to transactions but in addition include details on how to handle views

We dispense here with a presentation of DB-ASM rules and languages for views to give a detailed picture how AS^2 s can be specified on grounds of DB-ASMs; details were described in [26].

In [38], it has been shown that DB-ASMs capture exactly all database transformations on the same background. Thus, the sketched language will capture all AS^2 s.

2.4 Plots: High-Level Action Schemes for AS^2 s

In order to use a service (expressed as an AS^2), a sequence of service operations has to be executed. However, the sequencing of several service operations in order to execute a particular task is only left implicit in the AS^2 model. We now make it explicit by algebraic expressions called *plots*.

According to [37], a plot is a high-level specification of an action scheme, i.e. it specifies possible sequences of service operations in order to perform a certain task. For an algebraic formalisation of plots in WISs, it was possible to exploit KATs. Then a plot is an algebraic expression that is composed out of elementary operations including 0, 1 and propositional atoms, binary operators \cdot and $+$ and unary operators $*$ and $\bar{}$, the latter one being only applicable to propositions. With the axioms for KATs, we obtain an equational theory that can be used to reason about plots.

Propositions and operations testing them are considered the same. Therefore, propositions can be considered as operations, and overloading of operators for operations and propositions is consistent. In particular, 0 represents `fail` or `false`; 1 represents `skip` or `true`; $p \cdot q$ represents a sequence of operations or a conjunction, if both p and q are propositions; $p + q$ represents the choice between p and q or a disjunction, if both p and q are propositions; p^* represents iteration; and \bar{p} represents negation.

For our purposes here, the definition of plots for AS²s requires that we leave the purely propositional ground. The service operations give rise to *elementary processes* of the form

$$\varphi(\mathbf{x}) \text{ op}[\mathbf{z}](\mathbf{y}) \psi(\mathbf{x}, \mathbf{y}, \mathbf{z}),$$

in which op is the name of a service operation, \mathbf{z} denotes input for op selected from the view v with $\text{op} \in \text{Op}_v$, \mathbf{y} denotes additional input from the user and φ and ψ are first-order formulae denoting pre- and postconditions, respectively. The pre- and postconditions can be void, i.e. `true`, in which case they can be simply omitted. Furthermore, also simple formulae $\chi(\mathbf{x})$ —again interpreted as tests checking their validity—constitute elementary processes. With this we obtain the following definition.

Definition 16 The set of *process expressions* of an AS² is the smallest set \mathcal{P} containing all elementary processes that are closed under sequential composition \cdot , parallel composition \parallel , choice $+$ and iteration $*$. That is, whenever $p, q \in \mathcal{P}$ hold, then also pq , $p \parallel q$, $p + q$ and p^* are process expressions in \mathcal{P} .

The *plot* of an AS² is a process expression in \mathcal{P} .

Example 6 Let us look at some very simplistic examples. For a flight booking service, we may have the following (purely sequential) plot:

```
get_itineraries[](d) select_itinerary[i]()
  personal_data[](t) confirm_flight[](y)
    pay_flight[](c)
```

Here the parameters d, i, t, c and y represent dates, selected itinerary, traveller data, card details and a Boolean flag for confirmation.

Similarly, the following expression represents another plot for accommodation booking:

```
get_hotels[](d) select_hotel[h]()
  select_room[r]() personal_data[](t)
    confirm_hotel[](y) pay_accommodation[](c)
```

Here the parameters h and r represent the selected hotel and room.

Finally, the expression `personal_data[](t) (papers[](p) || discount[](d') payment[](c))` represents the plot of a conference registration service.

3 Service Ontologies: Describing Functional and Categorical Aspects of Services

As discussed in the introduction, the first purpose of a service ontology is to enable the location of services. For this, it is crucial that the service operations including the view defining queries and the plot of an AS² are provided with an adequate description, which will allow a search engine to discover (with some certainty) the required services. Such a description should comprise at least two parts:

- A *functional* description of input and output types as well as pre- and postconditions telling in technical terms what the service operation will do
- A *categorical* description by interrelated keywords telling what the service operation does by using common terminology of the application area.

According to [28], we normally require a third part addressing a *quality of service* (QoS) description of non-functional properties such as availability, response time, cost, etc., but the QoS description is not needed for service discovery, but for selection among alternatives. As discussed in the introduction, we consider QoS is part of a larger SLA description, which is related to the contracting aspect. We will discuss SLAs and QoS at the end of this section.

A functional description alone would be insufficient. For instance, a flight booking service operation requires an itinerary to be selected, so the input type could be specified as $\{\{flight_no : STRING, day : DATE, departure : TIME, class : CHAR, price : DECIMAL\}\}$, i.e. the input is a finite set of tuples, each of which defines a flight number, departure day and time, the booking class and the price. The output type could be similar with a status (confirmed, waitlisted, unavailable) added for each flight segment, i.e. we have the type $\{\{flight_no : STRING, day : DATE, departure : TIME, class : CHAR, price : DECIMAL, status : STRING\}\}$. A precondition could simply be that the selected itinerary is meaningful, i.e. flight numbers exist for the corresponding date and time and are compatible. However, no meaningful postcondition can be specified, as the output depends on the status of the (hidden) flight database. Moreover, a booking service for railway tickets would require the same types, so the functional description does not indicate exactly what kind of service is offered.

As for the categorical description, the terminology has to be specified. This defines an ontology in the widest sense, i.e. we have to provide definitions of “concepts” and relationships between them, such that each offered service becomes an instantiation of one or several concepts in the terminology. In this way we adopt the fundamental idea of the “semantic Web”. In the following, we will outline how description logics [5] can be exploited for service description.

As outlined above, the key to service discovery is a description of available services. Here we follow the already well-accepted approach to exploit description logics for this task. The reason for the use of description logics (since its very beginnings over 30 years ago) is that they enable the definition of concepts by necessary and sufficient conditions, and the logics are kept so simple that classification,

i.e. determining subsumption relationships, is decidable. Thus, a search requires a definition of the service sought by means of a complex concept. The well-known classification algorithms for description logics then can be used to determine all instances (in the ABox) matching the complex concept. As this is the standard, we do not repeat any of this in the paper.

3.1 Terminologies

As outlined, the functional, categorical and QoS description of services in a cloud requires the definition of an ontology. That is, we need a *terminological knowledge* layer (aka TBox in description logics) describing concepts and roles (or relationships) among them. This usually includes a subsumption hierarchy among concepts (and maybe also roles) and cardinality constraints. In addition, there is an *assertional knowledge* layer (aka ABox in description logics) describing individuals. Thus, services in a cloud constitute the ABox of an ontology, while the cloud itself is defined by the TBox.

In principle, instead of TBox and ABox, we could use the more classical notions of schema and instance and exploit any kind of data model. A query language associated with the used data model could then be used to find the required services. In fact, description logics only provide rather limited logics with respect to expressiveness. There are two major reasons for giving preference to description logics:

1. Description logics use two important relationships, which due to the restrictions become decidable: subsumption and instantiation. Subsumption is a binary relationship between concepts (denoted as $C_1 \sqsubseteq C_2$) guaranteeing that all instances of the subsumed concept C_1 are also instances of the subsuming concept C_2 . Instantiation defines a binary relationship between instances in the ABox and concepts in the TBox asserting that an element A of the ABox is an instance of a concept C in the TBox.

Subsumption and instantiation together allow us to discover services that are more expressive than needed but can be projected to a service just as required.

2. Concept and role names in the TBox can be subject to similarity search by a search engine. That is, the search engine could produce services that are similar (with a certainty factor) to the ones required with respect to the categorical description and match the functional description.

For simplicity, let us now look more closely into one particular description logic in the *DL-Lite* family (see [5]). For this, assume that C_0 and R_0 represent not further specified sets of basic concepts and roles, respectively. Then *concepts* C and *roles*

R are defined by the following grammar:

$$\begin{aligned} R &= R_0 \mid R_0^- \\ A &= C_0 \mid \top \mid \geq m.R \text{ (with } m > 0) \\ C &= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \end{aligned}$$

Definition 17 A *terminology* (or TBox) is a finite set \mathcal{T} of assertions of the form $C_1 \sqsubseteq C_2$ with concepts C_1 and C_2 as defined by the grammar above.

Each assertion $C_1 \sqsubseteq C_2$ in a terminology \mathcal{T} is called a *subsumption axiom*. Note that Definition 17 only permits subsumption between concepts, not between roles, though it is possible to define more complex terminologies that also permit role subsumption.

As usual, we use the shortcut $C_1 \equiv C_2$ instead of $C_1 \sqsubseteq C_2 \sqsubseteq C_1$. For concepts, \perp is a shortcut for $\neg\top$, and $\leq m.R$ is a shortcut for $\neg \geq m+1.R$.

Definition 18 A *structure* \mathcal{S} for a terminology \mathcal{T} consists of a non-empty set \mathcal{O} together with subsets $\mathcal{S}(C_0) \subseteq \mathcal{O}$ and $\mathcal{S}(R_0) \subseteq \mathcal{O} \times \mathcal{O}$ for all basic concepts R_0 and basic roles R_0 , respectively. \mathcal{O} is called the base set of the structure.

We first extend the interpretation of basic concepts and roles and to all concepts and roles as defined by the grammar above, i.e. for each concept C , we define a subset $\mathcal{S}(C) \subseteq \mathcal{O}$, and for each role R , we define a subset $\mathcal{S}(R) \subseteq \mathcal{O} \times \mathcal{O}$ as follows:

$$\begin{aligned} \mathcal{S}(R_0^-) &= \{(y, x) \mid (x, y) \in \mathcal{S}(R_0)\} \\ \mathcal{S}(\top) &= \mathcal{O} \\ \mathcal{S}(\geq m.R) &= \{x \in \mathcal{O} \mid \#\{y \mid (x, y) \in \mathcal{S}(R)\} \geq m\} \\ \mathcal{S}(\neg C) &= \mathcal{O} - \mathcal{S}(C) \\ \mathcal{S}(C_1 \sqcap C_2) &= \mathcal{S}(C_1) \cap \mathcal{S}(C_2) \\ \mathcal{S}(C_1 \sqcup C_2) &= \mathcal{S}(C_1) \cup \mathcal{S}(C_2) \\ \mathcal{S}(\exists R.C) &= \{x \in \mathcal{O} \mid (x, y) \in \mathcal{S}(R) \text{ for some } y \in \mathcal{S}(C)\} \\ \mathcal{S}(\forall R.C) &= \{x \in \mathcal{O} \mid (x, y) \in \mathcal{S}(R) \Rightarrow y \in \mathcal{S}(C) \text{ for all } y\} \end{aligned}$$

Definition 19 A *model* for a terminology \mathcal{T} is a structure \mathcal{S} , such that $\mathcal{S}(C_1) \subseteq \mathcal{S}(C_2)$ holds for all assertions $C_1 \sqsubseteq C_2$ in \mathcal{T} . A finite model, i.e. a model with a finite base set, is also called *instance* or ABox associated with \mathcal{T} .

Example 7 The general part of a service ontology could be defined by a terminology as follows:

$$\begin{aligned}
\text{Service} &\sqsubseteq \exists \text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists \text{address.URL} \sqcap \\
&\quad \exists \text{offered_by.Provider} \sqcap \leq 1.\text{address} \sqcap \leq 1.\text{offered_by} \\
&\quad \sqcap \exists \text{defining.Query} \sqcap \leq 1.\text{defining} \sqcap \exists \text{offers.Operation} \\
\text{Operation} &\sqsubseteq \exists \text{associated_with.Query} \sqcap \leq 1.\text{associated_with} \\
&\quad \text{Data_Service} \equiv \text{Query} \sqcap \geq 1.\text{defining}^- \\
&\quad \text{Functional_Service} \equiv \text{Operation} \sqcap \geq 1.\text{offers}^- \\
\text{Service_Operation} &\equiv \text{Data_Service} \sqcup \text{Functional_Service} \\
\text{Service_Operation} &\sqsubseteq \exists \text{input.Type} \sqcap \leq 1.\text{input} \\
&\quad \exists \text{output.Type} \sqcap \leq 1.\text{output} \\
\text{Type} &\sqsubseteq \exists \text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists \text{format.Format}
\end{aligned}$$

Here we used capital first letters to indicate concept names and lower-case letters for role names.

3.2 *Functional and Categorical Description*

As outlined above, we expect the terminology \mathcal{T} of a cloud to provide the functional, categorical and QoS description of its offered services.

The functional description of a service operation consists of input and output types as already indicated in Example 7 and pre- and postconditions. For the types, we need a type system with base types and constructors. For instance, the following grammar

$$\begin{aligned}
t &= b \mid \mathbb{I} \mid (a_1 : t_1, \dots, a_n : t_n) \mid \{t\} \mid [t] \mid \\
&\quad (a_1 : t_1) \oplus \dots \oplus (a_n : t_n)
\end{aligned}$$

describes (the abstract syntax of) a type system with a trivial type \mathbb{I} , a non-further specified collection of base types b and four-type constructors (\cdot) for record types, $\{\cdot\}$ for finite set types, $[t]$ for list types and \oplus for union types. Record and union types use field labels a_i .

The semantics of such types is basically described by their domain, i.e. sets of values $\text{dom}(t)$. Usually, for a base type b such as *Cardinal*, *Decimal*, *Float*, etc., the domain is some commonly known at most countable set with a common presentation. The domain of the trivial type contains a single special value, say

$dom(\mathbb{I}) = \{\perp\}$. For constructed types, we obtain the domain in the usual way:

$$\begin{aligned}
 dom((a_1 : t_1, \dots, a_n : t_n)) &= \\
 &\{(a_1 : v_1, \dots, a_n : v_n) \mid a_i \in dom(t_i) \text{ for } i = 1, \dots, n\} \\
 dom(\{t\}) &= \{A \mid A \subseteq dom(t) \text{ finite}\} \\
 dom([t]) &= \{[v_1, \dots, v_k] \mid v_i \in dom(t) \text{ for } i = 1, \dots, k\} \\
 dom((a_1 : t_1) \oplus \dots \oplus (a_n : t_n)) &= \bigcup_{i=1}^n \{(a_i : v_i) \mid v_i \in dom(t_i)\}
 \end{aligned}$$

In particular, a union type $(a_1 : \mathbb{I}) \oplus \dots \oplus (a_n : \mathbb{I})$ has the domain $\{(a_1 : \perp), \dots, (a_n : \perp)\}$, which can be identified with the set $\{a_1, \dots, a_n\}$, i.e. such types are in fact enumeration types.

It is no problem to add the specification of types to the general service terminology as outlined in Example 7, thereby defining part of the functional description.

Example 8 We can extend the terminology in Example 7 by the following axioms for types:

$$\begin{aligned}
 \text{Type} &\equiv \text{Base_type} \sqcup \text{Trivial_type} \sqcup \text{Composed_type} \\
 \text{Composed_type} &\equiv \text{Record} \sqcup \text{Set} \sqcup \text{List} \sqcup \text{Union} \\
 \text{Record} &\sqsubseteq \forall \text{component.Field} \\
 \text{Field} &\sqsubseteq \exists \text{field_name.Identifier} \sqcap \leq 1.\text{field_name} \\
 &\sqcap \exists \text{type.Type} \sqcap \leq 1.\text{type} \\
 \text{Union} &\sqsubseteq \forall \text{component.Field} \\
 \text{Record} \sqcap \text{Union} &\sqsubseteq \perp \\
 \text{Set} &\sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component} \\
 \text{List} &\sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component} \\
 \text{Set} \sqcap \text{List} &\sqsubseteq \perp
 \end{aligned}$$

Of course, the specification of composed types impacts directly on the format, which is defined by field names and the format for the component type(s). Nevertheless, this constraint can be handled by the specification of ABox assertions.

In addition to the types, the functional description of a service operation includes pre- and postconditions, which are defined by (first-order) predicate formulae. These formulae may contain further functions and predicates, which are subject to further (categorical) description.

Example 9 The terminology in Examples 7 and 8 can be further extended by the following axioms:

$$\begin{aligned}
 \text{Service_Operation} &\sqsubseteq \forall \text{pre.Condition} \sqcap \leq 1.\text{pre} \\
 &\sqcap \exists \text{post.Condition} \sqcap \leq 1.\text{post} \\
 \text{Condition} &\sqsubseteq \text{Formula} \sqcap \forall \text{uses.}(\text{Predicate} \sqcap \text{Function}) \\
 \text{Predicate} &\sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \neg \geq 1.\text{out} \\
 \text{Function} &\sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \exists \text{out.Type} \sqcap \leq 1.\text{out}
 \end{aligned}$$

This would complete the functional part of the terminology.

As shown at the beginning of this section, the functional description is insufficient for enabling service discovery, and the QoS description is only needed as a means to support the selection among several alternatives. The core of the service description by means of the terminology of a cloud is the categorical description, which refers to the standard terminology of the application area and relates the used notions to each other.

There are no general requirements for the categorical description, as it depends completely on the application domain. However, it will always lead to subconcepts of the concept *Service_Operation* plus additional concepts and roles. It will also add more details to the predicates and functions used in the pre- and postconditions.

Example 10 Let us look at booking services as required by a conference trip application with particular emphasis on flight booking. The categorical description may consist of the following axioms:

$$\begin{aligned}
 \text{Booking} &\sqsubseteq \text{Service_Operation} \sqcap \exists \text{initiator.Customer} \sqcap \\
 &\exists \text{initiated_by.Request} \sqcap \exists \text{receives.Acknowledgement} \\
 &\sqcap \exists \text{requires.Customer_data} \sqcap \exists \text{requires.Payment} \sqcap \\
 &\exists \text{receives.}(\text{Confirmation} \sqcup \text{Declination} \sqcup \text{Amendment}) \\
 \text{Request} &\sqsubseteq \exists \text{object.Booking_object} \sqcap \exists \text{date.DATE} \\
 \text{Flight_booking} &\sqsubseteq \text{Booking} \sqcap \forall \text{initiated_by.Flight_request} \\
 \text{Flight_request} &\sqsubseteq \text{Request} \sqcap \forall \text{object.Flight} \\
 \text{Flight} &\sqsubseteq \text{Booking_object} \sqcap \exists \text{number.Flight_number} \sqcap \\
 &\exists \text{carrier.Airline} \sqcap \exists \text{departure.Date} \sqcap \text{duration.Duration} \\
 &\sqcap \exists \text{origin.Airport} \sqcap \exists \text{destination.Airport}
 \end{aligned}$$

This specification is of course incomplete, but it shows how to proceed. That is, a booking is defined by a service operation that is initiated by a request from a customer; it further requires customer data and payment and leads to an acknowledgement plus a confirmation, declination or (suggested) amendment. The request for a booking contains at least a booking object—it could contain more than one—and a date. A flight booking is a booking that is initiated by a flight request, which is a request, in which all booking objects are flights. Flights themselves must have at least a flight number, an airline, a departure date, a duration and origin and destination airports.

3.3 *Service-Level Agreements*

SLAs have been widely discussed in the literature. By now, around 20 different types of SLAs have been identified [34], and depending on the viewpoint, these SLAs have been differently classified. Following our discussion in the introduction, we consider that the main purpose of SLAs is to determine as precisely as possible the rights and obligations that govern the relationship between service providers, service users and if applicable third parties. Technically, our approach consists of three parts:

- An extension of the service ontology describing the content of the SLAs
- A contracting framework that permits a contract skeleton to be extracted from the ontology
- A monitoring system that can be used to check when a violation to an SLA has occurred

As the service ontology is realised by some description logic, the contracting framework can be realised by queries against the ontology. This has been discussed by Rady using SPARQL to extract fragments of contracts from an SLA ontology [35]. We dispense with discussing this aspect any further in this chapter. Also, as stated in the introduction, SLA monitoring is still in an infant state, so we will not discuss it here, but we emphasise again that SLAs that cannot be monitored are de facto useless, i.e. *monitorability* of SLAs is a necessary property of services.

Different from the classification by Rady [34], we use the following classification schema, which puts a stronger emphasis on rights and obligations:

- *Terms of usage*: Some SLAs simply define general facts about the usage of services. Among these are the pricing schema, conditions for termination and suspension and the applicable jurisdiction. In particular, these facts do not require to be monitored. They will appear as part of the contract extracted from the ontology and can be used to check bills or determine legal actions in case of inaccuracies.

- *Technical aspects*: Some SLAs refer to technical properties of the services that are determined by the service model, i.e. they are not SLAs in the proper sense. These technical aspects cover two different areas:

- *Implementation aspects*: For instance, *portability* refers to the property of a service to be moved from one environment to another one, which is a property of the implementation. The same applies to *interoperability*, i.e. the property that the service can be combined with others; *scalability*, i.e. that the service can be applied to various input sizes; and *modifiability*, i.e. the property that the user may tune the service.

Furthermore, properties such as *testability*, *maintainability* and *verifiability* refer to software-technical characteristics. Actually, for a service user, it is much more important that a service has been adequately tested and verified, the results of these quality assurance measures are available and reproducible and preferably the service has already been certified according to some common quality standard rather than obtaining knowledge that verification, validation and testing can be done.

- *Usage aspects*: This is usually associated with a usability SLA. Terms such as understandability or learnability used in this context are only vaguely defined and thus cannot be used for monitoring purposes. Usability studies can nonetheless give recommendations to service users.

However, some of the technical aspects, in particular the implementation aspects, may also be regarded as defining obligations and commitments of the provider to guarantee particular features of the service, in which case the scope of the commitments made has to become part of the SLA.

- *Obligations and rights of the provider*: The most relevant class of SLAs covers obligations of the service provider, which also capture what the provider is committed to provide. The most commonly discussed SLAs in this class comprise the following:

- *Availability*: The SLA should cover when and for how long the service is guaranteed to be available to the user. Normally, this is formulated by some form of acceptable downtime. We will discuss availability SLAs further down.
- *Performance*: The SLA defines the expected (maximum/average) response time and throughput. Same as for availability, probability distributions could be used, but this is not state of the art.
- *Security and privacy*: SLAs concerning security and privacy could define the used methods for authentication, identity management, firewall rules, secrets to be preserved, confidentiality regulations, auditing procedures, etc. In our point of view, it appears advisable not only to register the obligations of the provider but also the means to be taken to ensure security and privacy.
- *Reliability*: This refers to the measures taken by the provider to ensure that message content and the service results as a whole are reliable.
- *Penalty and compensation*: These SLA captures mainly factual data about the amount to be repaid in case an SLA cannot be satisfied.

In addition, they may be SLAs capturing rights of the provider, e.g. to close down the service for maintenance or in case of imminent security threats—this could be coupled with alerting obligations—to update the service to a new version or release, to cancel a contract in parts or as a whole, to increase prices, etc.

- *Obligations and rights of the user*: Analogously, SLAs may refer to obligations of the users in particular with respect to usage and security/privacy regulations. These may also be subject to penalty and compensation regulations.

Example 11 Let us briefly look at SLAs for availability. In many current service offerings, these are rather vaguely formulated summing up small time intervals during which no external access is possible. Then 96.7% availability could mean a downtime of 2 s per minute or of 1 day per month, which depending on the user's needs makes a very big difference. Therefore, it may be better not to use only such coarse aggregates, but to rely on distributions. Part of an SLA ontology regarding availability could look as follows:

$$\begin{aligned}
 & \textit{Commitment} \sqsubseteq \textit{SLA} \quad \textit{AvailabilityCommitment} \sqsubseteq \textit{Commitment} \\
 & \textit{CommitmentValidity} \sqsubseteq \exists \textit{hasStart}.\#\textit{datetime} \sqcap \exists \textit{hasEnd}.\#\textit{datetime} \\
 & \quad \sqcap \textit{hasDurationEntity}.\textit{DurationEntity} \sqcap \leq 1.\textit{hasDurationEntity} \\
 & \textit{MaintenanceTime} \sqsubseteq \exists \textit{hasStart}.\#\textit{datetime} \sqcap \exists \textit{hasEnd}.\#\textit{datetime} \\
 & \quad \sqcap \textit{hasDurationEntity}.\textit{DurationEntity} \sqcap \leq 1.\textit{hasDurationEntity} \\
 & \quad \sqcap \textit{hasRepetition}.\textit{Repetition} \sqcap \leq 1.\textit{hasRepetition} \\
 & \textit{CommitmentValidity} \sqsubseteq \textit{AvailabilityCommitment} \\
 & \textit{MaintenanceTime} \sqsubseteq \textit{AvailabilityCommitment}
 \end{aligned}$$

This part covers the timing aspect, i.e. when the service will be available and when not. The second part covers how the availability is to be measured:

$$\begin{aligned}
 & \textit{ProbabilityDistribution} \sqsubseteq \textit{AvailabilityCommitment} \\
 & \textit{ProbabilityDistribution} \sqsubseteq \exists \textit{hasFormula}.\#\textit{string} \sqcap \exists \textit{hasParameter}.\#\textit{string} \\
 & \textit{MonitoringWindow} \sqsubseteq \textit{AvailabilityCommitment} \\
 & \quad \sqcap \textit{hasDurationEntity}.\textit{DurationEntity} \sqcap \leq 1.\textit{hasDurationEntity}
 \end{aligned}$$

A third part covers the recording of failures:

$$\begin{aligned}
 & \textit{ServiceRequest} \sqsubseteq \exists \textit{hasFailure}.\textit{ServiceFailure} \sqcap \exists \textit{hasRequest}.\#\textit{string} \\
 & \quad \sqcap \exists \textit{hasMonitoringWindow}.\textit{MonitoringWindow}
 \end{aligned}$$

$$\begin{aligned} & \sqcap \exists \text{hasDistribution.} \text{ProbabilityDistribution} \\ \text{ServiceFailure} & \sqsubseteq \exists \text{hasFailStr}\#string \sqcap \leq 1\text{hasFailStr} \end{aligned}$$

We dispense with extending the ontology with respect to refund commitments.

4 Service Mediation: Building Service-Centric Applications

In the introduction, we emphasised *mediation* as another important feature of a general theory of services on the Web. So let us now address the specification and instantiation of large-scale distributed systems exploiting services. This problem goes beyond service composition, even more beyond the composition of service operations. We will follow our previous work in [28].

One way to create such an application would be to start from a set of known services that are composed and extended by local components. This can be assumed to be well explored. The other way is to start from a specification of the composed specification, which can be taken as the plot of an AS². However, in this plot, we assume that most service operations are yet unknown; we only know a categorical description for them. For instance, some of the service operations may belong to a flight booking service, so we have to locate corresponding services using the service ontology of a service cloud and match them with the plot of the composed service. That is, besides search for services, we also need a notion of matching services.

The matching problem becomes particularly interesting, when we consider that services sought may be overlapping. For instance, when combining several booking services, each of them may contain a service operation for payment as well as one for gathering personal data. It would be not a very interesting composed application if such overlaps were not integrated. For the booking example, this would mean to have only one payment operation and gather personal data only once.

4.1 Service Mediators

With the concept of *service mediators*, we want to capture the plot of a composed AS². In other words, we want to define a plot of an application that is yet to be constructed. The key issue is that such mediators specify service operations to be searched for, which can then be used to realise the problem at hand in a service-oriented way.

In order to capture the idea to specify service requests, we relax the definition of a plot in such a way that service operations do not have to come from the same AS². Thus, in elementary processes, we use prefixes to indicate the corresponding AS², so we obtain $\varphi(\mathbf{x}) X : op[\mathbf{z}](\mathbf{y}) \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, in which X denotes a *service slot*, i.e. a placeholder for an actual service. Apart from this, we leave the construction

of the set of process expression as in Definition 16 with the only difference that also $\ell\text{-op}\langle p \rangle$ is a process expression, whenever p is one. Here, $\langle \cdot \rangle$ denotes a finite multiset constructor, i.e. we consider an arbitrary number of processes running in parallel, and $\ell\text{-op}$ denotes a multiset operation, which aggregates the query results of the different processes in the multiset.

Definition 20 A *service mediator* is a process expression with service slots. Furthermore, each service operation is associated with input and output types, pre- and postconditions and a concept in a service terminology.

Example 12 Let us specify a service mediator for a conference trip application, which should combine conference registration, flight booking and accommodation booking. Furthermore, replicative entry of customer data should be avoided, and confirmation of selection as well as payment should be unified in single local operations. This leads to the following specification:

```

personal_data[](t)
  (X : papers[](p) || X : discount[](d')
   ( union⟨Yj : get_itineraries[](d)⟩
     Yj : select_itinerary[i]() ||
     union⟨Zk : get_hotels[](d)⟩
     Zk : select_hotel[h]() Zk : select_room[r]() )
confirm[](y)
  (Yj : confirm_flight[](y) || Zk : confirm_hotel[](y))
pay[](c)
  (Yj : pay_flight[](c) || Zk : pay_hotel[](c) || X : payment[](c))

```

Here the slots X , Y_j and Z_k refer to services for conference registration, flight booking and accommodation booking, respectively, while the operations without prefix are considered to be local. For confirmation and payment, the input parameters y and c are simply pushed through to the two booking services.

4.2 Service Matching

A service mediator specifies which services are needed and how they are composed into a new plot of a composed AS². So we now need exact criteria to decide when a service matches a service slot in a service mediator.

It seems rather obvious that in such a matching criteria for all service operations in a mediator associated with a slot X , we must find matching service operations in the same AS², and the matching of service operations has to be based on their functional and categorical description. The guideline is that the placeholder in the mediator must be replaceable by matching service operations. Functionally, this means that the input for the service operation as defined by the mediator must be accepted by the matching service operation, while the output of the matching service operation must be suitable to continue with other operations as defined by

the mediator. This implies that we need supertypes and subtypes of the specified input and output types, respectively, in the mediator, as well as a weakening of the precondition and a strengthening of the postcondition. Categorically, the matching service operation must satisfy all the properties of the concept in the terminology that is associated with the placeholder operation, i.e. the concept associated with the matching service operation must be subsumed by that concept.

However, the matching of service operations is not yet sufficient. We also have to ensure that the projection of the mediator to a particular slot X results in a subplot of the plot of the matching AS^2 .

Definition 21 A *subplot* of a plot p is a process expression q such that there exists another process expression r such that $p = q + r$ holds in the equational theory of process expressions.

The *projection* of a mediator m is a process expression p_X such that $p_X = \pi_X(m)$ holds in the equational theory of process expressions, where $\pi_X(m)$ results from m by replacing all placeholders $Y : o$ with $Y \neq X$ and all conditions that are irrelevant for X by 1.

Based on this definition, it is tempting to require that the projection of a mediator should result in a subplot of a matching service. This would, however, be too simple, as order may differ and certain service operations may be redundant. We call such redundant service operations *phantoms*.

Definition 22 If for a condition $\varphi(\mathbf{x})$ appearing in a process expression p the equation $\varphi(\mathbf{x}) = \varphi(\mathbf{x})op[\mathbf{y}](\mathbf{z})$ holds, then $op[\mathbf{y}](\mathbf{z})$ is called a *phantom* of p .

That is, if the condition $\varphi(\mathbf{x})$ holds, we may execute the operation $op[\mathbf{y}](\mathbf{z})$ (or not) without changing the effect. Whenever $p = q$ holds in the equational theory of process expressions and $op[\mathbf{y}](\mathbf{z})$ is a phantom of p with respect to condition $\varphi(\mathbf{x})$, we may replace $\varphi(\mathbf{x})$ with $\varphi(\mathbf{x})op[\mathbf{y}](\mathbf{z})$ in q . Each process expression resulting from such replacements is called an *enrichment of p by phantoms*.

Thus, we must consider projections of enrichments by phantoms, which leads us to the following definition.

Definition 23 An AS^2 \mathcal{A} *matches* a service slot X in a service mediator m iff the following two conditions hold:

1. For each service operation $X : o$ in m , there exists a service operation op provided by \mathcal{A} such that:
 - The input type I_{op} of op is a supertype of the input type I_o of o .
 - The output type O_{op} of op is a subtype of the output type O_o of o .
 - $pre_o \Rightarrow pre_{op}$ holds for the preconditions pre_o and pre_{op} of o and op , respectively.
 - $post_{op} \Rightarrow post_o$ holds for the postconditions $post_o$ and $post_{op}$ of o and op , respectively.
 - The concept C_o associated with o in the service terminology subsumes the concept C_{op} associated with op .

2. There exists an enrichment m_X of m by phantoms such that building the projection of m and replacing all service operations $X : o$ by matching service operations op from \mathcal{A} result in a subplot of the plot of \mathcal{A} .

Example 13 Let us look again at the simple service mediator in Example 12. We can assume that the local operation $\text{personal_data}[](t)$ has the postcondition $\text{person}(t)$, and this is invariant under the service operations for itinerary and hotel selection. We can further assume that in both booking services, the service operation $\text{personal_data}[](t)$ is a phantom for $\text{person}(t)$. Thus, the mediator can be enriched by phantoms, which results in

```

personal_data[](t)
  (X : papers[](p) || X : discount[](d')
    (union{Y_j : get_itineraries[](d)
      Y_j : select_itinerary[i]() Y_j : personal_data[](t) ||
      union{Z_k : get_hotels[](d)
        Z_k : select_hotel[h]() Z_k : select_room[r]()
        Z_k : personal_data[](t)
      }
    }
confirm[](y)
  (Y_j : confirm_flight[](y) || Z_k : confirm_hotel[](y))
pay[](c)
  (Y_j : pay_flight[](c) || Z_k : pay_hotel[](c) || X : payment[](c))

```

The added phantom operations are highlighted. The projection of this process expression to the services X , Y_j and Z_k , respectively, results exactly in the three plots in Example 6.

4.3 Instantiation and Execution

Once matching services for all slots in a mediator have been found, we can build an instantiation of the mediator with real services, which serves as a high-level specification of a process that exploits several services.

Example 14 Consider again the mediator from Example 13. Suppose the slot X matches a conference registration service CONF_REG. Furthermore, let FL_BOOK and FLIGHT be two services matching the slot Y_j , while HOTEL_BOO is a matching hotel booking service for Z_k . Then the instantiated mediator becomes

```

personal_data[](t)
  (CONF_REG : papers[](p) || CONF_REG : discount[](d')
    (union{FL_BOOK : get_itineraries[](d), FLIGHT : get_itineraries[](d)
      FL_BOOK : select_itinerary[i]() FL_BOOK : personal_data[](t) || +
      FLIGHT : select_itinerary[i]() FLIGHT : personal_data[](t) ||
      HOTEL_BOO : get_hotels[](d)
        HOTEL_BOO : select_hotel[h]() HOTEL_BOO : select_room[r]()
        HOTEL_BOO : personal_data[](t)
      }
confirm[](y)
  ((FL_BOOK : confirm_flight[](y) + FLIGHT : confirm_flight[](y)) ||
    HOTEL_BOO : confirm_hotel[](y))

```

$$\text{pay}[](c) \\ ((\text{FL_BOOK} : \text{pay_flight}[](c) + \text{FLIGHT} : \text{pay_flight}[](c)) \parallel \\ \text{HOTEL_BOO} : \text{pay_hotel}[](c) \parallel \text{CONF_REG} : \text{payment}[](c))$$

Informally, this plot reads as follows. Start with gathering personal data t from a user of the composed service. This is a new operation performed locally. Then enter the service CONF_REG for conference registration. The first required service operation would be the entering of personal data, which can be done by passing on the already-collected data, so the interaction actually continues with two service operations $\text{papers}[](p)$ and $\text{discount}[](d')$ for entering papers p and any potential discount d' for the conference fee. These two service operations can be accessed in parallel. With this, the interaction with service CONF_REG is already finished.

We then enter (in parallel) the services FL_BOOK, FLIGHT and HOTEL_BOO for flight and hotel booking, respectively. For the first two, we first get itineraries using the service operation $\text{get_itineraries}[](d)$, which are combined by the union operator. We then select an itinerary i , which was provided either by FL_BOOK or FLIGHT. Depending on which service provided the selected itinerary, personal data are passed on to the service without involving the user, while the other service is no longer considered. Analogously, a hotel and a room in that hotel are selected using service operations $\text{select_hotel}[h]()$ and $\text{select_room}[r]()$, respectively, and again personal data are passed on.

The local operation $\text{confirm}[](y)$ would actually have to present the selections made and request confirmation y , which is then passed on to the corresponding service operations $\text{confirm_flight}[](y)$ and $\text{confirm_hotel}[](y)$ of FL_BOOK (or FLIGHT) and HOTEL_BOO, respectively.

Finally, the local operation $\text{pay}[](c)$ collects payment information and passes these on to the involved services, which then terminate.

It is clear from the definition of mediators by means of KAT expressions that an instantiated mediator is only a very high-level specification of a large-scale distributed application that runs several services at the same time. This becomes further evident by Example 14. Refining and implementing such a specification would require several add-ons. First, the involved services have to be started and terminated, which usually involves a login and authentication process. Then data has to be passed from the mediation process to the individual services, which bypass the user interaction, i.e. a control component associated with the process is needed. Furthermore, output from several services is combined, and a selection made by a user is passed back to the originating services, while non-selection leads to service termination. This must also be handled by the control component. That is, from the high-level specification of a composed application to an executable software, there is still some work to be done. However, the specification shows what is needed in the implementation.

Here, we used KATs to specify mediators, and thus, also instantiations of mediators result in KAT expressions. In [40], we discussed how to use ASMs instead or any other formalisms that allow us to specify processes. In particular, we could then exploit refinement in the same framework.

5 Conclusions

In this chapter, we reported on our continued research towards the fundamental question “what constitutes a (software) service (on the Web)”. We presented the *BDCM*² framework addressing the following important features of services:

- *Behaviour*: There must exist a general behavioural theory of services. For this we outlined our research on the model of Abstract State Services (AS²s) [26], which follows the line of research of the ASM thesis.
- *Description*: There must exist a description of a service that allows it to be discovered and used. For this we stressed service ontologies, e.g. the model in [27] addressing functional, categorical and quality aspects of services.
- *Contracting*: There must exist a contract between the service provider and user covering all relevant SLAs for the service. Here we outlined that the quality aspects of services should be extended to capture also all other aspects that could give rise to SLAs. The collection of SLAs has to be treated as a binding contract between the service provider and user.
- *Mediation*: A user must use a service in the contracted way but can build service mediations to realise service-centric applications satisfying his/her purposes. For this we presented our model of service mediators based on AS²s [28, 40].
- *Monitoring*: It must be possible to monitor the execution of a service in order to validate its behaviour and contracted SLAs. Here we outlined that research is still in an infant state, but a lot of research has been done already with respect to monitoring, though usually not in connection with SLAs.

The presented overview of our considerations concerning a general theory of services gives a snapshot of the status we have reached so far. In particular, behaviour can be captured by the behavioural theory underlying the AS² model, which also forms the basis for mediation. Description and contracting give rise to service ontologies capturing functional, categorical and contracting aspects. The first of these refers again to AS²s and the last one to SLAs. The first two aspects enable the location of services, which is also needed for matching mediator slots to available services. Finally, we claim that a service must always be coupled with a monitor that permits the SLAs to be verified at run time.

We believe that the *BDCM*² framework may serve as an umbrella for continuing the research, which is still not completed, in particular not for the aspects of contracting, mediation and monitoring. However, the framework has already taken up insights gained by others and rephrased them in a more general context. We welcome any critical comment concerning aspects that have been forgotten and contributions how to further sharpen the approach.

References

1. Akkiraju, R., et al.: Web service semantics: WSDL-S. <http://www.w3c.org/Submission/WSDL-S> (2005)
2. Alonso, G., et al. (eds.): *Web Services: Concepts, Architectures and Applications*. Springer, Berlin (2003)
3. Alves, A., et al.: Web services business process execution language, version 2.0. OASIS Standard Committee. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html> (2007)
4. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. *IBM Syst. J.* **47**(3), 377–396 (2008)
5. Baader, F., et al. (eds.): *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge (2003)
6. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. *Data Knowl. Eng.* **58**(3), 327–357 (2006)
7. Bergholtz, M., Andersson, B., Johannesson, P.: Abstraction, restriction, and cocreation: three perspectives on services. In: Trujillo, J., et al. (eds.) *Advances in Conceptual Modeling – Applications and Challenges*. Lecture Notes in Computer Science, vol. 6413, pp. 107–116. Springer, Berlin/Heidelberg (2010)
8. Bergholtz, M., Andersson, B., Johannesson, P.: Towards a model of services based on cocreation, abstraction and restriction. In: Jeusfeld, M.A., Delcambre, L.M.L., Ling, T.W. (eds.) *Conceptual Modeling – Proceedings of the 30th International Conference (ER 2011)*. Lecture Notes in Computer Science, vol. 6998, pp. 476–485. Springer, Berlin/Heidelberg (2011)
9. Bergholtz, M., Andersson, B., Johannesson, P.: Towards a model of services based on cocreation, abstraction and rights distribution. In: Thalheim, B., et al. (eds.) *Correct Software in Web Applications*. Springer, Vienna (2015, in this volume)
10. Blass, A., Gurevich, J.: Abstract state machines capture parallel algorithms. *ACM Trans. Comput. Log.* **4**(4), 578–651 (2003)
11. Bosa, K., Chelemen, R., Vleju, M.B.: A formal model of client-cloud interaction. In: Thalheim, B., et al. (eds.) *Correct Software in Web Applications*. Springer, Vienna (2015, in this volume)
12. Christensen, E., et al.: Web services description language (WSDL) 1.1. <http://www.w3c.org/TR/wSDL> (2001)
13. Erl, T.: *SOA: Principles of Service Design*. Prentice Hall Press, Upper Saddle River (2007)
14. Fensel, D., Bussler, C.: The web service modeling framework WSMF. *Electron. Commer. Res. Appl.* **1**(2), 113–137 (2002)
15. Fensel, D., et al.: *Enabling Semantic Web Services*. Springer, Berlin (2007)
16. Ferrario, R., Guarino, N., Fernández-Barrera, M.: Towards an ontological foundation for services science: the legal perspective. In: Sartor, G., Casanovas, P., Biasiotti, M., Fernández-Barrera, M. (eds.) *Approaches to Legal Ontologies. Law, Governance and Technology*, vol. 1, pp. 235–258. Springer, Netherlands (2011)
17. Geerts, G.L., McCarthy, W.E.: An ontological analysis of the economic primitives of the extended-REA enterprise information architecture. *Int. J. Account. Inf. Syst.* **3**(1), 1–16 (2002)
18. Gómez, J., Cachero, C., Pastor, O.: Modelling dynamic personalization in web applications. In: *Third International Conference on Web Engineering – ICWE 2003*. Lecture Notes in Computer Science, vol. 2722, pp. 472–475. Springer, Berlin/Heidelberg (2003)
19. Gurevich, J.: Sequential abstract state machines capture sequential algorithms. *ACM Trans. Comput. Log.* **1**(1), 77–111 (2000)
20. Hohfeld, W.N.: Fundamental legal conceptions as applied in legal reasoning. *Yale Law J.* **23**, 710–770 (1913)
21. Hruby, P.: *Model-Driven Design of Software Applications with Business Patterns*. Springer, New York (2006)
22. Keller, U., Lausen, H., Stollberg, M.: On the semantics of functional descriptions of web services. In: *Proceedings of the 3rd European Semantic Web Conference – ESWC 2006* (2006)

23. Lampesberger, H., Rady, M.: Monitoring of client-cloud interaction. In: Thalheim, B., et al. (eds.) *Correct Software in Web Applications*. Springer, Vienna (2015, in this volume)
24. Lusch, R.F., Vargo, S.L., Wessels, G.: Toward a conceptual foundation for service science: contributions from service-dominant logic. *IBM Syst. J.* **47**(1), 5–14 (2008)
25. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: Abstract state services. In: Song, I.Y., et al. (eds.) *Advances in Conceptual Modeling – Challenges and Opportunities, ER 2008 Workshops*. *Lecture Notes in Computer Science*, vol. 5232, pp. 406–415. Springer, Berlin/Heidelberg (2008)
26. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: A theory of data-intensive software services. *SOCA* **3**(4), 263–283 (2009)
27. Ma, H., Schewe, K.D., Wang, Q.: An abstract model for service provision, search and composition. In: Kirchberg, M., et al. (eds.) *Services Computing Conference - APSCC 2009*, pp. 95–102. *IEEE Asia Pacific* (2009)
28. Ma, H., Schewe, K.D., Thalheim, B., Wang, Q.: A formal model for the interoperability of service clouds. *SOCA* **6**(3), 189–205 (2012)
29. McCarthy, W.E.: The REA accounting model: a generalized framework for accounting systems in a shared data environment. *Account. Rev.* **57**(3), 554–578 (1982)
30. O’Sullivan, J., Edmond, D., Ter Hofstede, A.: What is a service? Towards accurate description of non-functional properties. *Distrib. Parallel Databases* **12**(2–3), 117–133 (2002)
31. Papazoglou, M.P., van den Heuvel, W.J.: Service-oriented design and development methodology. *Int. J. Web Eng. Tech.* **2**(4), 412–442 (2006)
32. Papazoglou, M.P., van den Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. *VLDB J.* **16**(3), 389–415 (2007)
33. Preist, C.: A conceptual architecture for semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen F. (eds.) *The Semantic Web – ISWC 2004*. *Lecture Notes in Computer Science*, vol. 3298, pp. 395–409. Springer, Berlin/Heidelberg (2004)
34. Rady, M.: Parameters for service level agreements generation in cloud computing: a client-centric vision. In: Castano, S., et al. (eds.) *Advances in Conceptual Modeling – ER 2012 Workshops*. *Lecture Notes in Computer Science*, vol. 7518, pp. 13–22. Springer, Berlin/Heidelberg (2012)
35. Rady, M.: Generating an excerpt of a service level agreement from a formal definition of non-functional aspects using owl. *J. Univers. Comput. Sci.* **20**(3), 366–384 (2014)
36. Sampson, S.E., Froehle, C.M.: Foundations and implications of a proposed unified services theory. *Prod. Oper. Manag.* **15**(2), 329–343 (2006)
37. Schewe, K.D., Thalheim, B.: Conceptual modelling of web information systems. *Data Knowl. Eng.* **54**(2), 147–188 (2005)
38. Schewe, K.D., Wang, Q.: A customised ASM thesis for database transformations. *Acta Cybernetica* **19**(4), 765–805 (2010)
39. Schewe, K.D., Wang, Q.: A formal model for service mediators. In: Trujillo, J., et al. (eds.) *Advances in Conceptual Modeling - Applications and Challenges (ER 2010 Workshops)*. *Lecture Notes in Computer Science*, vol. 6413, pp. 76–85. Springer, Berlin/Heidelberg (2010)
40. Schewe, K.D., Wang, Q.: Preferential refinements of abstract state machines for service mediators. In: Muccini, H., Tang, A. (eds.) *Proceedings of QSIC 2012*, pp. 158–166. *IEEE CPS, Xi’an* (2012)
41. Simple Object Access Protocol (SOAP): <http://www.w3c.org/TR/soap>
42. Stollberg, M., Cimpian, E., Mocan, A., Fensel, D.: A semantic web mediation architecture. In: *Proceedings CSWWS 2006* (2006)
43. Universal Description, Discovery and Integration (UDDI): <http://www.uddi.org>
44. Web Ontology Language (OWL): <http://www.w3c.org/OWL/>
45. Zeithaml, V.A., Parasuraman, A., Berry, L.L.: Problems and strategies in services marketing. *J. Mark.* **49**(2), 33–46 (1985)

Codesign of Web Information Systems

Bernhard Thalheim and Klaus-Dieter Schewe

Abstract Web information systems are nowadays widely used. E-business, education, infotainment, community and identity Web systems are data and information intensive. They integrate a variety of database, workflow and other processing, communication and presentation systems. Their design and development is thus based on an integrated development of structuring, functionality, distribution and interactivity of users. The codesign framework allows the integration of these different aspects. This chapter surveys the codesign approach and its deployment for the development of large Web information systems.

1 Introduction

1.1 *The Path from Database Systems to Web Information Systems*

Database design is based on one or more database models. Often, design is restricted to structural aspects. Static semantics, which is based on static integrity constraints, is sparsely used. Processes are then specified after implementing structures. Behaviour of processes can be specified by dynamic integrity constraints. Views for interaction are defined on top of the structuring and functionality. This approach is the classical *local-as-view* schema design approach.

Information systems aim at delivering information to the user. Users have their own specific *information demands*. It depends on the tasks the user has to perform and on the skills and abilities of users, i.e. user portfolio and user profile. The user thus demands data in the right form, the right format and the

B. Thalheim

Department of Computer Science, Christian Albrechts University Kiel, 24098 Kiel, Germany

e-mail: thalheim@is.informatik.uni-kiel.de

<http://www.is.informatik.uni-kiel.de/~thalheim>

K.-D. Schewe (✉)

Software Competence Centre Hagenberg, 4232 Hagenberg, Austria

e-mail: kd.schewe@scch.at

<http://www.scch.at/de/schewe-klaus-dieter>

© Springer International Publishing Switzerland 2015

B. Thalheim et al. (eds.), *Correct Software in Web Applications and Web Services*,

Texts & Monographs in Symbolic Computation,

DOI 10.1007/978-3-319-17112-8_9

right size and structuring and at the right moment and under consideration of the user’s information demand, similar to online analytical processing (OLAP) or data warehouse processing.

Web information systems (WIS) add, however, two more aspects: distribution and presentation. Web systems are typically highly distributed and use a variety of systems, e.g. an ensemble of database systems. There is no need for integration of these systems into a singleton system. Distribution can be specified on the basis of services and collaboration features, such as exchange frames. Therefore, we have to develop a *global-as-view* approach for Web information systems. Presentation systems and information systems are bundled together based on some architecture, e.g. client-server architectures. Information is delivered to the presentation system in such form that the system can appropriately lay out the data and support the user by functionality.

1.2 Abstraction Layers in Web Information System Development

System engineering on the basis of the triptych consisting of the application domain description at the strategic level, the requirement prescriptions at the business user level and finally the systems specifications for the information system and the presentation system are considered in [4] and [12]. The specification is oriented towards systems that are easy and intuitive to use. The methodology for the development of Web information systems is based on an *abstraction layer model*, which is illustrated in Fig. 1a. Web information systems have two different faces:

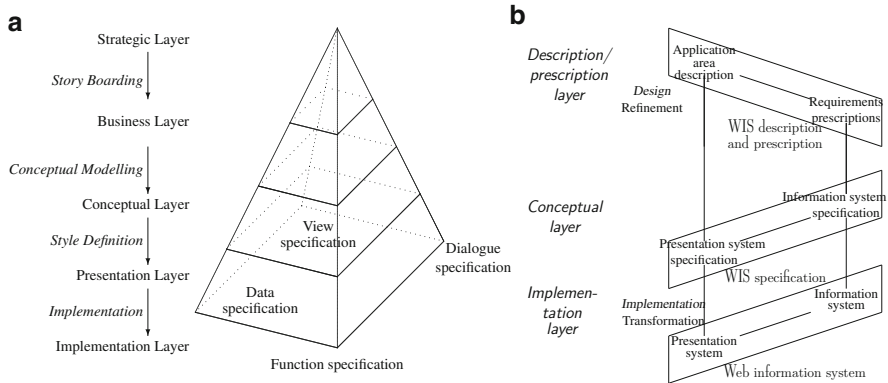


Fig. 1 Abstraction layers in Web information systems. (a) The abstraction layer model. (b) The dichotomy in WIS development

the systems perspective and the user perspective. These perspectives are tightly related to each other. We consider the presentation system to be an integral part of WIS. It satisfies all user requirements. It is based on real-life cases [28]. The dichotomy is displayed in Fig. 1b where the right side represents the system perspective and the left side of the ladder represents the user perspective.

1.3 The Six Concerns of Web Information Systems Specification

We consider six different concerns for Web information systems: intention, usage, content, functionality, context and presentation.

Intention: The intention aspect is a very general one centred around a mission statement for the system. The primary question is what is the purpose of the system?

Usage: Once some clarity with respect to the intentions of the Web-based system has been obtained, the question arises by whom and how the system will be used. As Web-based systems are open systems, it is important to anticipate the behaviour of the users. We model usage through storyboards that allow the specification of the stories users might use with the Web information system.

Content: The content aspect concerns the question: which information should be provided? It is coupled with the problem of designing an adequate database. However, the organisation of data that is presented to the user via a website is significantly different from the organisation of data in a database.

Functionality: The functionality aspect is coupled with the question of whether the site should be passive or active. A passive site only allows a user to navigate through the pages without any activity. In an active site, information is also required from the user. Specific functions allow the processing of user input and the provision of features such as searching, printing, marking and extraction.

Context: The context aspect deals with the context of the Web information system with respect to society, time, expected users, the history of utilisation and the paths of these users through the system.

Presentation: The presentation aspect concerns the final realisation by Web pages. This depends on the support of technical end devices such as computer screens, televisions, cell phones, etc., and set layout preferences.

The six concerns can be mapped to conceptual structures that are used for Web information system specification:

1. We start with the characteristics used for the strategic layer. The main specification elements used are intention and mission. They are mapped to metaphors, general goals, rhetorical figures and patterns and grids of Web pages discussed later.

2. The scenarios reflect the utilisation by actors, for which we envision a number of stories that correspond to real use. These scenarios may be captured through observation of reality. Story spaces and plots are recorded in various levels of detail through the methods discussed in [25]. The stories are reflected in the storyboard.
3. Content specification is the basis for the media types, i.e. data types and their functions, which will be introduced in part III. It combines data specification with user requirements and is reflected in the content portfolio.
4. Functionality is provided by the media types as required by the storyboard. The typical standard functions are navigation, retrieval (search), support functions and feedback facilities.
5. Context is based on tasks, history and environment. We use the specification of context for restructuring and functionality enhancement, which will form the basis of EXtensible Stylesheet Language (XSL) transformations, and the onion approach that is discussed in the last part of the paper.
6. Presentation depends on the intention, the provider, the technical environment available and the users that the WIS is targeting. Presentation results in the layout and the playout of the WIS. *Layout* requires the development of multimedia presentations for each page. *Playout* additionally requires the development of functionality that supports visits of users depending on the story they are currently following to achieve their goals. Layout and playout integrate the chosen metaphors; they depend on chosen page patterns and grids as well as on quality requirements.

Conceptual structures and their association are depicted in Fig. 2. We may separate the syntactics and pragmatics layers. Arrows represent associations of the kind ‘uses’, ‘part-of’ or ‘relates’. For instance, the story is based on the user and the functions. Information metaphors relate content to information.

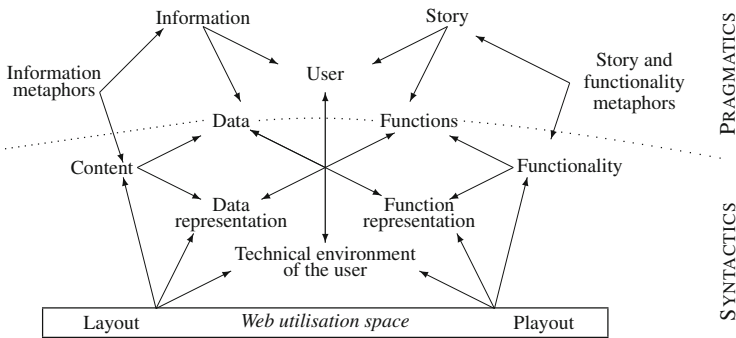


Fig. 2 The Web utilisation space based on the WIS concerns

1.4 *Web Information Systems Design and Development*

The problem of Web information system design can be thus stated as follows: Design the logical and physical structure of a Web information system for given database management systems (or for a database paradigm) and given presentation systems, so that it contains all the information required by the user and required for the efficient behaviour of the whole information system for all users. Furthermore, specify the application processes and the user interaction.

The implicit goals of Web information system design are the following:

- To meet all the information (contextual) requirements of the entire spectrum of users in a given application area
- To provide a “natural” and easy-to-understand structuring of the information content
- To preserve the designers’ entire semantic information for a later redesign
- To achieve all the processing requirements and also a high degree of efficiency in processing
- To achieve logical independence of query and transaction formulation on this level
- To provide a simple user interface family that is easy to comprehend

Database design has been extensively discussed in the past.¹ Almost all outstanding questions have been extensively discussed. Modelling, design and development of Web information systems include, however, further aspects:

Structuring of a Web information system application is concerned with representing the database structure and the corresponding static integrity constraints.

Functionality of a Web information system application is specified on the basis of processes and dynamic integrity constraints.

Distribution of Web information system components is specified through explicit specification of services and exchange frames.

Interactivity is provided by the system on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

This understanding has led to the **codesign approach** to modelling by specification *structuring, functionality, distribution and interactivity*. These four aspects of modelling, design and development have both syntactic and semantic elements. Additionally, they have pragmatic elements, since a broad variety of users has to be considered.

¹See, for instance, the large bibliography in [32].

1.5 Survey on Codesign for Web Information Systems Design

Web information systems specification, development and design can be based on extended entity-relationship models, on models that specify distribution and on models for behaviour and interactivity description on the user's side. Structuring uses extended entity-relationship models. It is based on hierarchical predicate logic. Functionality is defined based on a higher-order entity-relationship model (HERM) algebra, on query forms and transactions and on VisualSQL. Interactivity is declared in an integrated form based on SiteLang that allows description of dialogue scenes, stories, story space with actors and scenario. Distribution can be defined through a service specification and exchange frames. These models can be handled in a coherent way. There are method compilations of the design into other models.

Constructs of the codesign languages are:

- Structuring is given by the pair (Structure, StaticConstraints).
- Functionality is specified by the pair [(StateChange∪Retrieval)Operations, DynamicConstraints]. Operations are specified on the basis of HERM algebra (for modification and retrieval) which provides a language for generalised views and which can be enhanced to media types.
- Distributivity is declared by service (informational process, service manager, competence) and exchange frame (architecture collaboration style, collaboration pattern). Services can be declared on the basis of generalised views (media types).
- Interactivity is specified by the story spaces, actors, media objects and the presentation. The story space is a graph of scenes and activities.

2 Codesign of Schema-Centric Database Systems: The Local-as-View Approach

Database systems are based on the *local-as-view* approach. Functionality is defined in an algebra that uses the structure of the database system. Create, retrieval (queries), update and delete (CRUD) functions are algebraic expressions. Views are defined by queries. They support users and satisfy their data demand and their activities with the database system.

The entity-relationship (ER) model was introduced by P.P. Chen in 1976 [5]. The model conceptualises and graphically represents the structure of the relational model. It is currently used as the main conceptual model for database and information system development. Due to its extensive usage, a large number of extensions to this model were proposed in the 1980s and 1990s. Cardinality constraints [5, 10, 11, 19, 32] are the most important generalisation of relational database constraints [31]. These proposals have been evaluated, integrated or

explicitly discarded in the intensive research discussion surrounding this area. The semantic foundations proposed in [9, 14, 32] and the various generalisations and extensions of the entity-relationship model have led to the introduction of the higher-order or hierarchical entity-relationship model [32], which integrates most of the extensions and also supports conceptualisation of functionality, distribution [33] and interactivity [25] for information systems. Class diagrams of the unified modelling language (UML) standard are a special variant of extended entity-relationship models.

The higher-order entity-relationship model (HERM)² is a language for defining the structure (and functionality) of database or information systems. Its structure is developed inductively. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed. Entity types conceptualise structuring of things of reality through attributes. Cluster types generalise types or combine types into singleton types. Relationship types are associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined for a type. Typical integrity constraints of the extended entity-relationship model are participation, look-across and general cardinality constraints. Entity, cluster and relationship classes contain a finite set of objects defined on these types. The types of a HERM schema are typically depicted by a HERM diagram.

The main application area for extended ER models is the conceptualisation of database applications. Database schemata can be translated to relational, XML or other schemata based on transformation profiles that incorporate properties of the target systems. The ER model has had a deep impact on the development of diagramming techniques in the past and still influences extensions of the unified modelling language (UML). UML started with binary relationship types with look-across constraints and without relationship-type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow for distinguishing generalisation from specialisation.

2.1 Languages for Structure Specification

Structuring of databases is based on three interleaved and dependent parts:

Syntactics: *Inductive specification of structures* uses a set of base types, a collection of constructors and a theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. **Structural recursion** is the main specification vehicle.

²To be more precise, the higher-order entity-relationship modelling language.

Semantics: *Specification of admissible databases* on the basis of static integrity constraints describes those database states which are considered to be legal. If structural recursion is used, then a variant of hierarchical first-order predicate logics may be used for description of integrity constraints.

Pragmatics: *Description of context and intension* is based either on explicit reference to the enterprise model, to enterprise tasks, to enterprise policy and to environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location and common sense.

The classical four-layered approach is used for inductive specification of database structures. The first layer is the data environment, called the basic data type scheme, which is defined by the system or is the assumed set of available basic data types. The second layer is the schema of a database. The third layer is the database itself, representing a state of the application's data often called micro-data. The fourth layer consists of the macro-data that are generated from the micro-data by application of view queries to the micro-data.

2.1.1 An Example of a HERM Diagram

The HERM schema uses a formal language for schema definition and diagrams for graphical representation of the schema. Let us consider an infotainment Web information system. An element of this system is the event database. Events are

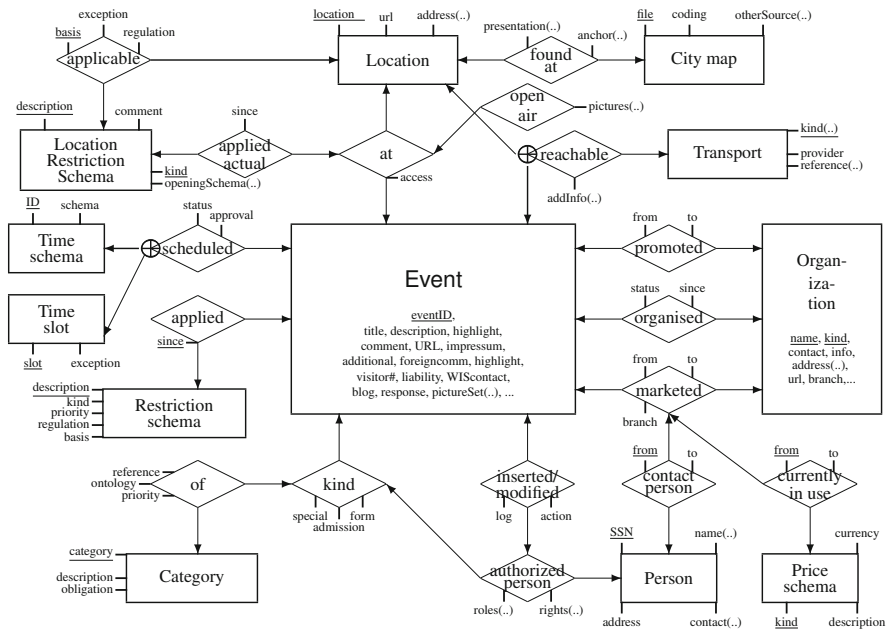


Fig. 3 Event database schema in an infotainment application

characterised by their title, description, comments, etc. Their identification must be enhanced by a surrogate key eventID if we are not using very complex identification. Events can be separated into different kinds and have some special additional information. Events are categorised depending on their kind. Organisations promote, organise or market these events. People are involved in the marketing and in the maintenance of the database. Events are organised at certain locations at a certain time. The event database also includes legal restrictions, transportation information and other data that are injected from other databases.

Attributes are identified by the type that uses the attribute. Therefore, we may neglect the unique name assumption. Attributes are typically structured. For instance, the name of a person is given by a sequence of first names, a family name, a set of academic titles and potentially a family title, e.g.

name(firstNames <firstName>, famName, [acadTitles{aTitle}], [familyTitle]).

Entity types are represented graphically by rectangles. Attributes primarily identifying a type are underlined. We may use attribute types outside the corresponding rectangle or diamond. Another association is to include attributes inside the type. Relationship types are represented graphically by diamonds and associated by directed arcs to their components. A cluster type is represented by a diamond, is labelled by the disjoint union sign and has directed arcs from the diamond to its component types. Alternatively, the disjoint union representation \oplus is attached to the relationship type that uses the cluster type. In this case, directed arcs associate the \oplus sign with component types. An arc may be annotated with a label.

2.1.2 The Definition Scheme for Structures in HERM

The extended entity-relationship model uses a data type system for its attribute types. It allows the construction of entity types $E \triangleq (\text{attr}(E), \Sigma_E)$ where E is the entity type defined as a pair—the set $\text{attr}(E)$ of attribute types and the set Σ_E of integrity constraints that apply to E . The definition *def* of a type T is denoted by $T \triangleq \text{def}$.

The HERM schema lets users inductively build relationship types $R \triangleq (T_1, \dots, T_n, \text{attr}(R), \Sigma_R)$ of order i ($i \geq 1$) through a set of (labelled) types of order less than i , a set of attribute types and a set of integrity constraints that apply to R . The types T_1, \dots, T_n are the components of the relationship type. Entity types are of order 0. Relationship types are of order 1 if they only have entity types as component types. Relationship types are of order i if all component types are of order less than i and if one of the component types is of order $i - 1$.

Additionally, cluster types $C \triangleq T_1 \dot{\cup} \dots \dot{\cup} T_n$ of order i can be defined through a disjoint union $\dot{\cup}$ of relationship types of order less than i or of entity types.

Entity/relationship/cluster classes T^C contain a set of objects of the entity/relationship/cluster type T . The HERM schema mainly uses set semantics, but (multi-) list or multi-set semantics can also be used. Integrity constraints apply to their type and restrict the classes. Only those classes are considered for which the constraints

of their types are valid. The notions of a class and of a type are distinguishable. Types describe the structure and constraints. Classes contain objects.

The data type system is typically inductively constructed on a base type B by application of constructors such as the tuple or products constructor $(..)$, set constructor $\{..\}$ and the list constructor $\langle .. \rangle$. Types may be optional component types and are denoted by $[..]$.

The types T can be labelled $l : T$. The label is used as an alias for the type. Labels denote roles of the type. Labels must be used if the same type is used several times as a component type in the definition of a relationship or cluster type. In this case, they must be unique.

An entity-relationship schema consists of a set of data, attribute, entity, relationship and cluster types where they are types that are inductively built on the basis of the base types.

Given a base-type system B , the types of the ER schema are defined through the type equation:

$$T = B \mid (l_1 : T, \dots, l_n : T) \mid \{T\} \mid \langle T \rangle \mid [T] \mid T \dot{\cup} T \mid l : T \mid N \stackrel{\circ}{=} T$$

2.2 Static Integrity Constraints

Integrity constraints are used to separate “good” states or sequences of states of a database system from those which are not intended. Constraints are given by users at various levels of abstraction, with a variety of vagueness and intensions behind them and on the basis of different languages.

Each structure is also based on a set of *implicit model-inherent integrity constraints*:

Component-construction constraints are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.

Identification constraints are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups [2]. We shall see later that value representability or weak-value representability leads to controllable structuring. Acyclicity and finiteness of structuring support axiomatisation and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

Superficial structuring leads to representation of constraints through structures. In this case, implication of constraints is difficult to characterise.

Implicit model-inherent constraints belong to performance and maintenance traps. Often, names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal

semantics allows us to derive synonym, homonym, antonym, troponym, hypernym and holonym associations among the constructs used.

Integrity constraints can be specified based on the Beeri-Vardi (BV) frame, i.e. by an implication with a formula for premises and a formula for the implication. BV constraints do not lead to rigid limitation of expressibility. If structuring is hierarchical, then BV constraints can be specified within first-order predicate logic. We may introduce a variety of different classes of integrity constraints, defined as such:

Equality-generating constraints allow to generate equalities among objects or components of objects for a set of objects from one class or from several classes.

Object-generating constraints require the existence of another object set for a set of objects satisfying the premises.

A class C of integrity constraints is called *Hilbert-implication-closed* if it can be axiomatised by a finite set of bounded derivation rules and a finite set of axioms. It is well known that the set of joint dependencies is not Hilbert-implication-closed for relational structuring. However, an axiomatisation exists with an unbounded rule, i.e. a rule with potentially infinite premises.

The most important class of integrity constraints of the HERM schema is the class of cardinality constraints. Other classes of importance for the HERM schema are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[31]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints. Multivalued dependencies can directly be represented by ER structures since they separate concerns [34]. Classical cardinality constraints are participation constraints, look-across constraints and general cardinality constraints.

The diagram in Fig. 3 can be enhanced by an explicit representation of cardinality and other constraints. If participation constraints $card(R, R') = (m, n)$ are used for a component consisting of one type R' , then the arc from R to R' is labelled by (m, n) (see Fig. 5). If look-across constraints $look(R, R') = m..n$ are used for binary relationship types, then the arc from R to R' is labelled by $m..n$.

2.3 Representation Alternatives

The classical approach to database objects is to store an object based on strong typing. Each real-life thing is thus represented by a number of objects which are either coupled by the object identifier or supported by specific maintenance

procedures. In general, however, we might consider two different approaches to representation of objects:

Class-wise, identification-based representation: things of reality may be represented by several objects. The *object identifier* (OID) supports identification without representing the complex real-life identification. Objects can be elements of several classes. In the early days of object orientation, it has been assumed that objects belong to one and only one class. This assumption has led to a number of migration problems which have no satisfactory solutions. Structuring based on extended ER models [32] or object-oriented database systems uses this option. Technology of relational and object-relational database systems is based on this representation alternative.

Object-wise representation: graph-based models which have been developed in order to simplify the object-oriented approaches [2] display objects by their subgraphs, i.e. by the set of nodes associated to a certain object and the corresponding edges. This representation corresponds to the representation used in standardisation.

XML is based on object-wise representation. It allows the use of null values without notification. If a value for an object does not exist, is not known, is not applicable or cannot be obtained, the XML schema does not use the tag corresponding to the attribute or the component. Classes are hidden.

Object-wise representation has a high redundancy which must be maintained by the system, thus decreasing performance to a significant extent. Besides the performance problems, such systems also suffer from low scalability and an insufficient utilisation of resources. The operating of such systems leads to lock avalanches. Any modification of data requires a recursive lock of related objects.

For these reasons, object-wise representation is applicable only under a number of restrictions:

- The application is stable, and the data structures and the supporting basic functions necessary for the application are not changed during the lifespan of the system.
- The data set is almost free of updates. Updates, insertions and deletions of data are only allowed in well-defined restricted “zones” of the database.

Typical application areas for object-wise storage are archiving systems, information presentation systems and content management systems. They use an update system underneath. We call such systems **playout systems**. The data are stored in the same way in which they are transferred to the user. The data modification system has a **playout generator** that materialises all views necessary for the playout system. Other applications are main-memory databases without updates. The SAP database system uses a huge set of related views.

We may use the first representation for our *storage engine* and the second representation for the *input engine* or the *output engine* in data warehouse approaches.

2.4 HERM Schemata

The schema is based on a set DD of base (data) types which are used as value types for attribute types. A set $\{E_1, \dots, E_n, C_1, \dots, C_l, R_1, \dots, R_m\}$ of entity, cluster and (higher-order) relationship types on base data types DD is called *schema* \mathcal{S} if the relationship and cluster types use only the types from $\{E_1, \dots, E_n, C_1, \dots, C_l, R_1, \dots, R_m\}$ as components and cluster and relationship types are properly layered.

A HERM schema is defined by the pair $\mathcal{D} = (\mathcal{S}, \Sigma)$ where \mathcal{S} is a schema and Σ is a set of constraints defined on \mathcal{S} . A database \mathcal{D}^C on \mathcal{D} consists of classes for each type in \mathcal{D} such that the constraints Σ are valid.

The classes of the extended ER model have been defined through sets of objects on the types. In addition to sets, lists, multi-sets or other collections of objects may be used. In this case, the definitions used above can easily be extended [32].

A number of domain-specific extensions have been introduced to the ER model. One of the most important is the extension of the base types by spatial data types such as point, line, oriented line, surface, complex surface, oriented surface, line bunch and surface bunch. These types are supported by a large variety of functions such as meets, intersects, overlaps, contains, adjacent, planar operations and a variety of equality predicates.

The translation of the schema to (object-)relational or XML schemata can be based on a profile [32]. Profiles define which translation choice is preferred over other choices, how hierarchies are treated, which redundancy and null-value support must be provided, which kind of constraint enforcement is preferred, which naming conventions are chosen, which alternative for representation of complex attributes is preferred for which types and whether weak types can be used. The treatment of optional components is also specified through the translation profile of the types of the schema. A profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.

2.5 Operations for Information Systems

The higher-order entity-relationship model uses an inductive structuring. This inductive structuring can also be used for the introduction of functionality. Functionality specification is based on HERM algebra and can easily be extended to HERM/QBE, VisualSQL, query forms and transactions. HERM algebra uses type-preserving functions and type-creating functions.

General operations on type systems can be defined by *structural recursion*. Given types T, T' and a collection type C^T on T (e.g. set of values of type T , bags, lists) and operations such as generalised union \cup_{C^T} , generalised intersection \cap_{C^T} and

generalised empty elements \emptyset_{C^T} on C^T . Given further an element h_0 on T' and two functions defined on the types $h_1 : T \rightarrow T'$ and $h_2 : T' \times T' \rightarrow T'$.

Then we define the structural recursion by insert presentation for R^C on T as follows: $srec_{h_0, h_1, h_2}(\emptyset_{C^T}) = h_0$ $srec_{h_0, h_1, h_2}(\{\!\|s\|\!\}) = h_1(s)$ for singleton collections $\{\!\|s\|\!\}$ $srec_{h_0, h_1, h_2}(\{\!\|s\|\!\} \cup_{C^T} R^C) = h_2(h_1(s), srec_{h_0, h_1, h_2}(R^C))$ iff $\{\!\|s\|\!\} \cap_{C^T} R^C = \emptyset_{C^T}$.

All operations of the object-relational database model, of the extended entity-relationship model and of other declarative database models can be defined by structural recursion, e.g.

- Selection is defined by $srec_{\emptyset, \iota_\alpha, \cup}$ for the function

$$\iota_\alpha(\{o\}) = \begin{cases} \{o\} & \text{if } \{o\} \models \alpha \\ \emptyset & \text{otherwise} \end{cases}.$$

- (Natural) join is defined by $\bowtie = srec_{\emptyset, \bowtie_T, \cup}$ for the type $T = T_1 \times T_2$, the function

$$\bowtie_T(\{(o_1, o_2)\}) = \{o \in \text{Dom}(T_1 \cup T_2) \mid o|_{T_1} = o_1 \wedge o|_{T_2} = o_2\}$$

and the type $T' = C^{T_1 \cup T_2}$.

- Aggregation functions can be defined based on the two functions for null values

$$h_f^0(s) = \begin{cases} 0 & \text{if } s = \text{NULL} \\ f(s) & \text{if } s \neq \text{NULL} \end{cases}$$

$$h_f^{\text{undef}}(s) = \begin{cases} \text{undef} & \text{if } s = \text{NULL} \\ f(s) & \text{if } s \neq \text{NULL} \end{cases}$$

through structural recursion, e.g.

$$\text{sum}_0^{\text{null}} = srec_{0, h_{1d}^0, +} \text{ or}$$

$$\text{sum}_{\text{undef}}^{\text{null}} = srec_{0, h_{1d}^{\text{undef}}, +};$$

$$\text{count}_1^{\text{null}} = srec_{0, h_1^0, +} \text{ or}$$

$$\text{count}_{\text{undef}}^{\text{null}} = srec_{0, h_1^{\text{undef}}, +}$$

or the doubtful SQL definition of the average function

$$\frac{\text{sum}_0^{\text{null}}}{\text{count}_1^{\text{null}}}.$$

Similarly we may define intersection, union, difference, projection, join, nesting and un-nesting, renaming, insertion, deletion and update. Operations may be either used for retrieval of values from the database or for state changes within the database. A *HERM query* is simply an expression of HERM algebra.

2.6 *Dynamic Integrity Constraints*

Database dynamics is defined on the basis of transition systems. A transition system on the schema S is a pair

$$\mathcal{TS} = (\mathcal{S}, \{\overset{a}{\longrightarrow} \mid a \in \mathcal{L}\})$$

where \mathcal{S} is a non-empty set of state variables, \mathcal{L} is a non-empty set (of labels) and $\overset{a}{\longrightarrow} \subseteq \mathcal{S} \times (\mathcal{S} \cup \{\infty\})$ for each $a \in \mathcal{L}$.

For the transition system \mathcal{TS} , we introduce a *temporal dynamic database logic* using the quantifiers \forall_f (always in the future), \forall_p (always in the past), \exists_f (sometimes in the future) and \exists_p (sometimes in the past).

The most important class of dynamic integrity constraints are *state-transition constraints* $\alpha O \beta$ which use a precondition α and a post-condition β for each operation O . The state-transition constraint $\alpha O \beta$ can be expressed by the temporal formula $\alpha \overset{O}{\longrightarrow} \beta$. Each finite set of static integrity constraints can be equivalently expressed by a set of state-transition constraints $\{\wedge_{\alpha \in \Sigma} \alpha \overset{O}{\longrightarrow} \wedge_{\alpha \in \Sigma} \alpha \mid O \in \text{Alg}(M)\}$.

Integrity constraints may be enforced

- Either at the procedural level by application of
 - Trigger constructs [18] in the active event-condition-action setting
 - Greatest consistent specialisations of operations [23]
 - Stored procedures, i.e. fully fledged programs considering all possible violations of integrity constraints
- Or at the transaction level by restricting sequences of state changes to those which do not violate integrity constraints
- Or by the DBMS on the basis of declarative specifications depending on the facilities of the DBMS
- Or at the interface level on the basis of consistent state changing operations

2.7 *Specification of Workflows*

A large variety of approaches to workflow specification has been proposed in the literature. We prefer formal descriptions with graphical representations that thus avoid pitfalls of methods that are entirely based on graphical specification such as the “and/or” traps. A *basic computation step algebra*, introduced in [35], may be used.

Instead we can also use the business process modelling and notation (BPMN) language [36]. This language separates users and their roles, e.g. the editor pool with the separation into swimlanes such as the *release session* and the *request to*

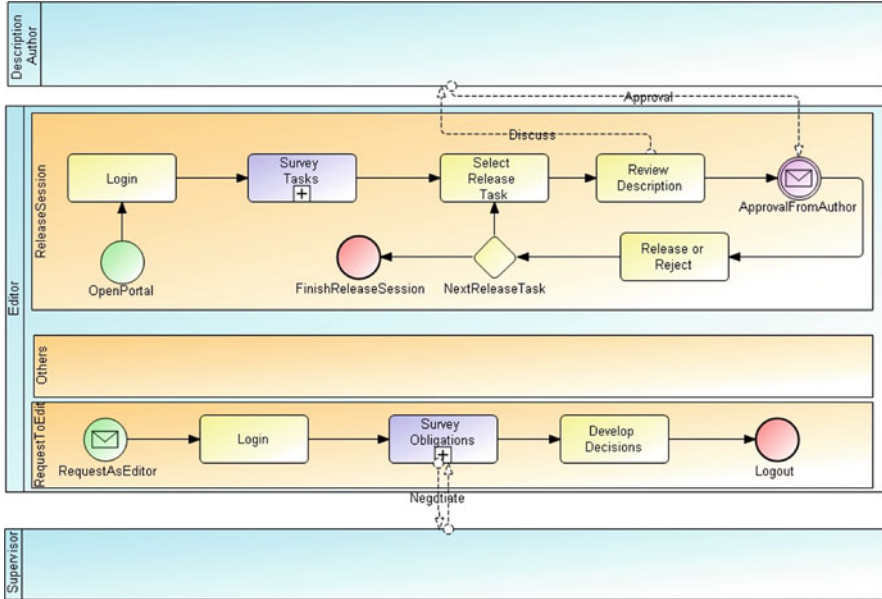


Fig. 4 A BPMN diagram for the editing process of an infotainment website

edit in Fig. 4. BPMN rigorously describes the syntactical and graphical elements as they are used by business analysts and operators to define and control the business activities (operations on data) and their (event or process-driven and possibly resource-dependent) execution order. In BPMN, a business process is represented by a diagram or graph where nodes are executed and where arcs are used to contain and pass the control or execution order information. The activities are performed in a certain order, depending on resources being available, data or control conditions to be true and events to happen. The main elements of diagrams are events (depicted by circles), activities (rounded rectangles) and gates (diamonds) for exclusive, conjunctive, disjunctive or event-controlled split or join. Business processes representing roles may exchange messages (dotted arrow) with other roles.

BPMN is a language that supports business process developers in a positive way, yet hinders them at the same time; the “principle of linguistic relativity” [37] postulates that actors skilled in a language may not have a (deep) understanding of some concepts of other languages.

Processes are built by separation of the specification into workflow process and workflow process instances. A singleton isolatable process instance is bound to its control token. Inter-process collaboration is supported exclusively through messages and events. Resource dependence is hidden. Swimlanes correspond to different roles of users. Pools are used for views on process sets. Nodes in a diagram are separated into activity, event and control flow (called gates) nodes. Events are

either boundary (start, end) or intermediate events. Tasks comprehend only some of possible executions ($\{ \text{Service, User, Receive, Send, Script, Manual, Reference, None} \}$). The rigid localisation in diagrams imposes context sensitivity of functions, avalanches of side constraints, none-incremental semantics (e.g. goto jumps) and resulting orchestration problems.

2.8 Views in the Local-As-View Approach

The HERM schema can be used to define views, e.g. the view schema in Fig. 5. A singleton view is defined by a query that maps the HERM schema to new types. Combined views also may be considered which consist of singleton views which together form another HERM schema.

A view schema³ is specified over a HERM schema \mathcal{D} by a schema $\mathcal{V} = \{S_1, \dots, S_m\}$, an auxiliary schema \mathcal{A} and a (complex) query $q : \mathcal{D} \times \mathcal{A} \rightarrow \mathcal{V}$ defined on \mathcal{D} and \mathcal{A} . Given a database \mathcal{D}^C and the auxiliary database \mathcal{A}^C , the view is defined by $q(\mathcal{D}^C \times \mathcal{A}^C)$.

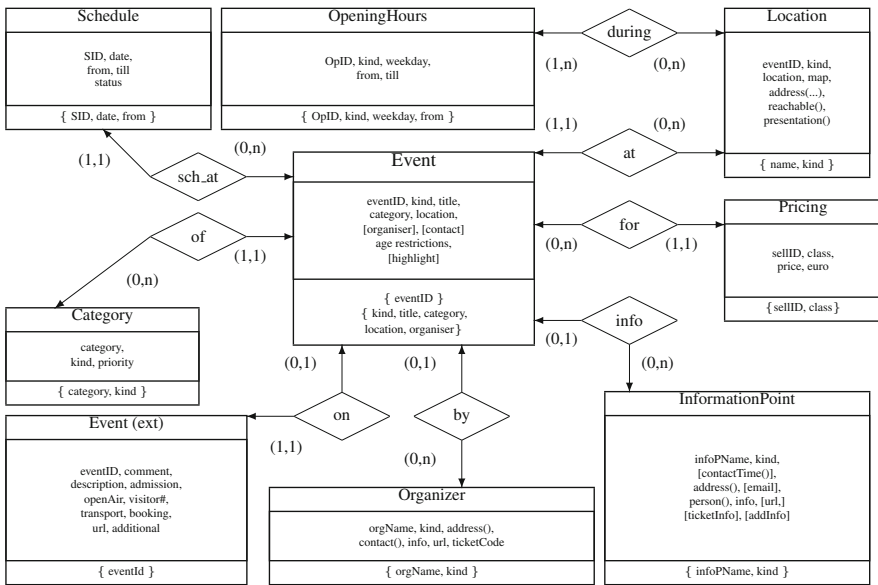


Fig. 5 View schema defined on the schema in Fig. 3

³Views are typically defined as a schema. This extension of the relational approach where views are single-table views is necessary for the maintenance of coherence within a view.

Views can be inductively defined on the schemata generated so far. Thus, *view towers* may be defined. View towers support the reuse of views within other view definitions. Typical views in a view tower are:

1. Security views for database protection
2. Viewpoint views
3. Data manipulation views and data retrieval views

A view tower is a suite of views that are inductively defined. A first-layer view is defined on the database schema. An (i+1)-layer view is defined on views of lower layers and on the database schema. The view definition is enhanced by a maintenance policy. Views can be materialised and enhanced by functions.

Views can have redundant elements, e.g. the attribute *eventID*. In Fig. 5, we also use two different keys for the type *Event*. A *view suite* consists of a set of views, an integration or association schema and obligations requiring maintenance of the association. The integration is defined over the view schemata. Obligations are based on the master-slave paradigm, i.e. the state of the view suite classes is changed whenever an appropriate part of the database is changed. Views are used to deliver *content* to the user.

Additionally, views should support services. Services request information and features from the service provider. The classical approach links information to content and features to functions. We generalise the view schema by the frame for content and functions:

```

generate MAPPING :  VARS → OUTPUT STRUCTURE
  from DATABASE TYPES
  where SELECTION CONDITION
  represent using GENERAL PRESENTATION STYLE
    & ABSTRACTION (GRANULARITY, MEASURE, PRECISION)
    & ORDERS WITHIN THE PRESENTATION  & POINTS OF VIEW
    & HIERARCHICAL REPRESENTATIONS  & SEPARATION
  browsing definition CONDITION & NAVIGATION
  functions SEARCH FUNCTIONS & EXPORT FUNCTIONS & INPUT FUNCTIONS
    & SESSION FUNCTIONS & MARKING FUNCTIONS
    
```

These generalised view schemata are the basis for *service specification* [1] (see Fig. 6). They can be decomposed to relational views and are thus supported by (object-)relational technology. We may develop retrieval views and data maintenance views as well as security views. Additionally, auxiliary views are used for support of insert, delete and update functions for those views that are not directly updateable.

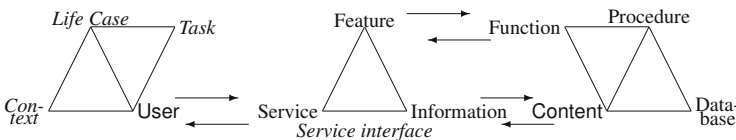


Fig. 6 Service architectures for Web information systems

3 Codesign of Socio-Technical Systems: Database Services in the Task-Centred Approach

The design of information systems is often system-centric. Such systems tend to be unserviceable, non-parsimonious, overly complex and require learning efforts from the user. Instead, a user seeks information. The notion of information is often given in a syntactic (e.g. entropy based), semantic (e.g. non-derivable data) or business-oriented (e.g. useful data) form. However, we prefer the anthropomorphic (or pragmatic) notion of the concept of information: *Information as processed by humans, is*

- *data perceived or noticed, selected and organised by its receiver,*
- *because of his/her subjective human interests,*
- *originating from his/her instincts, feelings, experience, intuition, common sense, values, beliefs, personal knowledge, or wisdom*
- *simultaneously processed by his/her cognitive and mental processes, and*
- *seamlessly integrated in his recallable knowledge.*

An information system is thus based on a database system that provides the data to the user. These data are views on the databases.

This notion makes it possible to specify the information demand of a user. The *information demand* of a user depends on the tasks the user has to perform and on the skills and abilities of users, i.e. user portfolio and user profile. The user thus demands data in the right form, the right format and the right size and structuring, at the right moment and in dependence on user's tasks and circumstances. The *activity demand* of a user also depends on the portfolio of a user, i.e. on a space of tasks the user has to perform or may perform, on the corresponding obligations and permissions and on the roles the user is playing in his/her world.

Figure 6 displays the “Janus” head of socio-technical systems. The user world is driven by life cases, tasks and context. The information system world is composed of a database system, on views defined on top of the database, on procedures which are supported by the database management system and on functions which support the user work. The service interface is the mediating connector that allows the user to satisfy his/her information and activity demand. This demand depends on the support needed, e.g. for workplace and workspace requested, for data consumed or produced by the user and for the environment and context of the user. The user is characterised by a profile, e.g. the work profile, the education profile and the personality profile.

The mediating service can be supported by *media objects* which are specified through *media types* [24]. They provide the content and functions in such a form that information and features of a service directly be associated with the content and the functions.

3.1 *Application- and User-Driven Design of Systems*

Users need information systems to provide a solution for their tasks in dependence on their life cases. Therefore, we characterise the user viewpoint by a profile of the user, by a task portfolio and by their life cases in Fig. 6. For task completion, users need the right kind of data, at the right time, in the right granularity and format, unabridged and within the frame agreed upon in advance. Moreover, users are bound by their ability to verbalise and digest data and their habits, practices and cultural environment. To avoid intellectual overburdening of users, we must first observe real applications before the system development leading to life cases [29]. Life cases specify the concrete life situation of the user and thus characterise a bundle of tasks that the user should solve. Syntax and semantics of life cases have already been well explored in [26].

User modelling is based on the specification of *user profiles* that address the characterisation of the users and the specification of *user portfolios* that describe the users' tasks and their involvement and collaboration on the basis of the mission of the Web information system [26].

To characterise the users of a Web information system, we distinguish between *education*, *work* and *personality* profiles. The education profile contains properties that the users can obtain by education or training. Capabilities and application information which have come about as a result of educational activities are also suitable for this profile. Properties will be assigned to the work profile if they can be associated with task solving information and skills in the application area, i.e. task expertise and experience as well as system experience. Another part of a work profile is the interaction profile of a user, which is determined by the frequency, intensity and style of utilisation of his/her Web information system. The personality profile characterises the general properties and preferences of a user. General properties are the status in the enterprise, community, etc., and the psychological and sensory properties like hearing, motor control, information processing and anxiety.

A *portfolio* is determined by responsibilities and is based on a number of targets. Therefore, the actor portfolio (referring to *actors* as groups of users with similar behaviour) within an application is based on a set of tasks assigned to or intended by an actor and for which she/he has the authority and control and a description of involvement within the task solution [27]. A *task* as a piece of work is characterised by a problem statement, initial and target states, collaboration and presupposed profiles, auxiliary conditions and means for task completion. Tasks may consist of subtasks. Moreover, the task execution model defines what, when, how, by whom and with which data a task can be accomplished. The result of executing a task should present the final state as well as the satisfaction of target conditions.

3.2 *Services that Satisfy the User Demand*

Although services are developed, used, applied and intensively discussed in modern practice, the concept of an information service has not yet been introduced. Services are artefacts that can be utilised by many users in different contexts at different points of time in different locations and serve a certain purpose. This notion generalises the REA (resource-event-agent) framework. A service system can be specified [6] through the classical rhetorical frame introduced by Hermagoras of Temnos [quis, quid, quando, ubi, cur, quem ad modum, quibus adminiculis (W7: who, what, when, where, why, in what way, by what means)]. Services are primarily characterised by W4: wherefore (end), whereof (source), wherewith (supporting means) and worthiness [(surplus) value]. Additionally, the purpose can be characterised by answering the why, whereto, when and for which reason W4 questions. The secondary characterisation W14H is given by characterising the user or stakeholder (by whom, to whom, whichever), the application domain (wherein, where, for what, wherefrom, whence, what), the solution they are providing (how, why, whereto, when, for which reason) and the additional context (whereat, whereabout, whither, when).

An information service provides information and features. Features allow the user to use the data in the specific form and activity. Typical features are navigation, structuring, query/answer interfaces, export/import support, sessions and marking. The *service interface* is thus the mediator between the user and the system worlds.

3.3 *Task-Centred Development for Database Systems as a Service*

Information systems support a large variety of users that have different educational, work and personality profiles. These users perform tasks in dependence on their life cases. These tasks can be specified as goal- or purpose-oriented actions. Tasks can be structured into subtasks that are restricted by conditions, data and context, e.g. organisations, policies, environment and channel. Tasks form the portfolio of a user.

Task specification is *usage centred*. Description of tasks includes identifying essential user and business purposes, understanding targeted users by roles in relation to site, understanding tasks in terms of user intentions and needs, prioritising user roles and user tasks in terms of expected frequency and business importance and engineering the design of a technical system to fit user and business priorities. Therefore, participatory design relates tasks to services.

Modern information system engineering uses the triptych consisting of the application domain description, the requirements prescriptions and finally the systems specifications [4, 13]. The main methods for application domain description are life case specification, intention description, profile and portfolio description and sufficient understanding of context. It describes the application domain as it is or

as it should be without any reference to systems. The database system serves as a service in the task-centred approach.

3.4 Database and Knowledge Base Systems that Support Services

A database system that supports services supports the user by views on the data in the database and by functions. These views deliver information to the user and accept the data of the user. Functions support the features requested by the user. Functions can be complex and are in this case small database programs, i.e. procedures defined in the database management system languages. Views are typically based on queries defined on the database system.

The views and the functions are combined into the media object. In a nutshell, a media object is defined by an extended view on some underlying database, which can then serve for provision of information and features of an elementary scene. Media objects are defined over media types and can be combined into containers that deliver or stream them to the user within a Web layout system.

The database system may consist of several database systems. This combination is necessary whenever a monolithic system cannot be developed. The infotainment system in Fig. 7 delivers information to visitors, inhabitants and interested users.

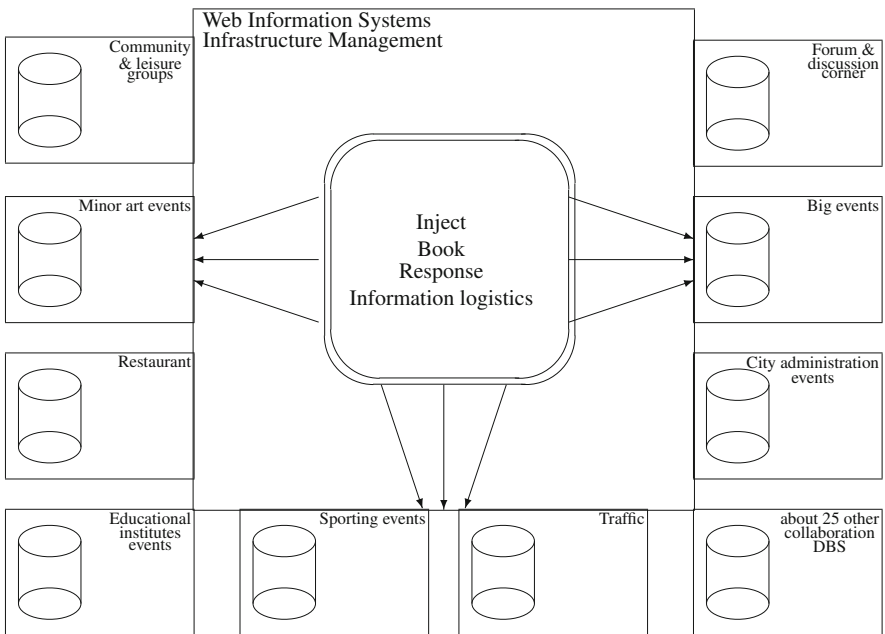


Fig. 7 The architecture of a database system providing views and functions for infotainment sites

Infotainment systems are one kind of Web information systems. They are more information intensive and mainly support information services. Identity websites are another type of a Web information system that is typically less information intensive. Community, edutainment and e-business Web information systems satisfy the activity demand of a user as well.

4 Codesign of Distributed Database Systems: The Global-As-View Approach to Collaboration

Specification of distribution has been neglected for a long period of time. Instead of explicit specification of distribution, multi-database systems and federated database systems have been extensively discussed in the literature. On the other hand, database research has succeeded in developing approaches that incorporate conceptual specification and allow to reason on systems at a far higher abstraction level. With the advent of Web information systems, systems became naturally distributed.

The information society changed with the advent of the Web. Nowadays, applications have become highly distributed. In the past, systems were developed on the whole on the basis of paradigms of programming in the large. This approach still considers systems to be holistic. Monolithic programming is going to be changed to *programming in the Web*:

- Flexible, scalable and adaptable communication with a variety of protocols and exchange frames
- Runtime coordination of cooperating partners based on contracts, obligations, permissions and tolerated deviations
- Exclusive or shared database components
- Separation agreement for input capsules, output capsules and untouchables
- Exchange architectures for collaboration depending on policy, profile, pattern and style

4.1 Collaboration of Distributed Systems

Distributed systems use services of local database. These services are defined by views on the local database systems. They provide their own data and functionality. These systems collaborate. Collaboration can be specified in the 3C approach by their three aspects (3C framework):

Communication is defined via exchange of messages and information or classically defined via services and protocols [17]. It depends on the choice of media, transmission modes, meta-information, conversation structure and paths and on the restriction policy.

Coordination is specified via management of individuals and their activities and resources. It is the dominating perspective of collaboration. The specification is based on the pre-/post-articulation of tasks and on the description management of tasks, objects and time. Coordination may be based on loosely or tightly integrated activities and may be enabled, forced or blocked.

Cooperation is the production taking place on a shared space. It can be considered as the workflow or life case perspective. We may use a specification based on storyboard-based interaction [30] that is mapped to (generic and structured) workflows. The information exchange is based on media types for production, manipulation and organisation of contributions.

The *collaboration* style is based on four components describing:

Supporting programs of the information system including session management, user management and payment or billing systems

Data access pattern for data *release* through the net, e.g. broadcast or P2P, for *sharing* of resources either based on transaction, consensus and recovery models or based on replication with fault management, and for *remote access* including scheduling of access

The style of collaboration on the basis of peer-to-peer models or component models or push-event models which restrict possible communication

The coordination workflows describing the interplay among parties, discourse types, name space mappings and rules for collaboration

Collaboration pattern supports access and configuration (wrapper facade, component configuration, interceptor, extension interface), event processing (reactor, proactor, asynchronous completion token, accept connector), synchronisation (scoped locking, strategic locking, thread-safe interface, double-checked locking optimisation) and *parallel execution* (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage):

Proxy collaboration, which uses partial system copies (remote proxy, protection proxy, cache proxy, synchronisation proxy, etc.).

Broker collaboration, which supports coordination of communication either directly, through message passing, based on trading paradigms, by adapter-broker systems or callback-broker systems.

Master/slave collaboration, which uses tight replication in various application scenarios (fault tolerance, parallel execution, precision improvement, as processes, threads, with(out) coordination).

Client/dispatcher collaboration, based on name spaces and mappings.

Publisher/subscriber collaboration, also known as the observer-dependent paradigm. It may use active subscribers or passive ones. Subscribers have their subscription profile.

Model/view/controller collaboration, similar to the three-layer architecture of database systems. Views (see Sect. 2.8) and controllers define the interfaces.

4.2 *Architectures for Distribution*

A number of *architectures* have already been proposed in the past for massively distributed and collaborating systems. In the sequel, we use the 3C (or 3K) model for specification of distribution and collaboration. Collaboration will be supported on the basis of exchange frames and information service [20]. The first specify dissemination, e.g. announcement, volume, time and protocols. The latter are used for specification of the information service with extraction, transformation, load and representation. Such distributed services are based on classical communication facilities such as routing, e.g. P2P (as in the case with query based network propagation), such as large nets, and partially closed subnets and propagation.

4.3 *Coordination Specification and Contracts*

Communication and cooperation is nicely supported in the classical setting by communication systems and workflow systems. Coordination specification is, however, still a research problem. Coordination supports the consistency of work products and of work progress and is supported by an explicitly specified coordinator. If work history is of interest, a version manager is integrated into the exchange support system. The coordination is supported by an infrastructure component. The coordination component observes modification of data that are of common interest to collaborating parties and resolves potential conflicts. The conflict resolution strategy is based on a cooperation contract. The contract is global to all parties and may contain extensions for peer-to-peer collaboration of some of the parties.

The coordinator is based on description of contracts. They describe who collaborates with whom, who supports it, in which scenario or story on which topic and on which (juridical) basis. Coordination is based on a coordination contract. The contract consists of:

- The coordination party characterisation and their roles, rights and relations;
- The organisation frames of coordination specifying the time and schema, the synchronisation frame, the coordination workflow frame and the task distribution frame;
- The context of coordination;
- The quality requirements (ubiquity, security, interpretability, consistency, view consistency, scalability, durability, robustness, performance) for coordination.

We can distinguish four levels of coordination specification. The syntactical level uses an interface description language (IDL) description and may use coordination constructs of programming languages. We use constructs of the JDL (job description language) for this description of resources, obligations, permissions and restrictions. The behaviour level specifies failure-atomicity, execution-atomicity, pre, rely, guarantee and post conditions and preconditions. The synchronisation level

specifies service object synchronisation and paths and maintains a synchronisation counter or monitor. The fourth level specifies a quality of services level. The coordination profile is specified by a coordination contract, coordination workspace, synchronisation profile, coordination workflow and task distribution.

We distinguish between the frame for coordination and the actual coordination. Any actual coordination is an instance of the frame. Additionally, it uses an infrastructure. The contract specifies the general properties of coordination. Several variants of coordination may be proposed. The formation of a coordination may be based on a specific infrastructure. For instance, the washer may provide a workspace and additional functionality to the collaborating parties.

4.4 Exchange Frames for Distribution

Exchange frames might be specified through the triple

(architecture, collaboration style, collaboration pattern).

The exchange architecture usually provides a system architecture integrating the information systems through communication and exchange systems. The collaboration style specifies the supporting programs, the style of cooperation and the coordination facilities. The collaboration pattern specifies the roles of the parties and their responsibilities and rights and the protocols that they may rely on. Distributed database systems are based on local database systems and follow a certain integration strategy. Integration is based on total integration of the local conceptual schemata into a global distribution schema.

Figure 7 illustrates an architecture of a distributed database system. It uses local databases with export facilities as masters in a publish-subscribe collaboration pattern, a master database as the central kernel for information delivery to different users, for customer management and for integrated information display to different users in an infotainment website.

Besides the classical distributed system, we support also other architectures such as *database farms* in Fig. 8, *incremental information system societies* and *cooperating information systems*. Incremental information system societies are the basis for facility management systems. Simple incremental information systems are data warehouses and content management systems. The exchange architecture may include the workplace of the client describing the parties, groups or organisations, roles and rights of parties within a group, the task portfolio and the organisation of the collaboration, communication and cooperation.

With the advent of Web information systems, we face the *heterogeneity trap*: presentation systems of users follow a completely different paradigm and system culture. We thus have to extend architectures and exchange frames and services for such systems.

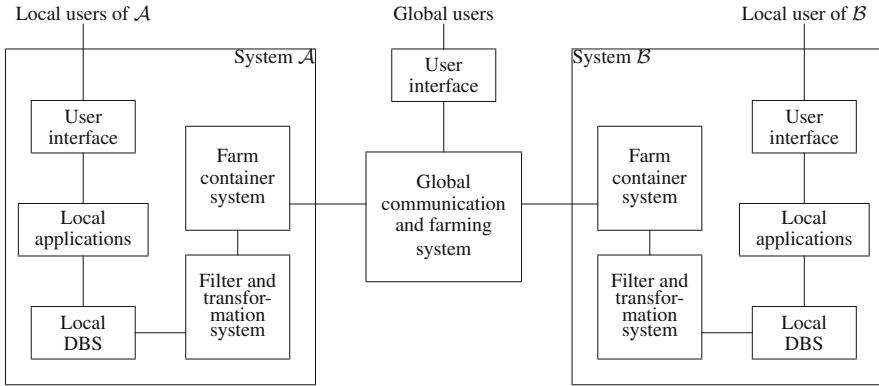


Fig. 8 An architecture of database systems farm

5 Storyboarding for Presentation Systems: The Usage-Driven Approach

Web information systems must support life cases for a variety of users. These life cases are typically combined into application stories that describe how a system is supposed to be used. Naturally, a conceptual model for application stories must be centred around the users: what do they do and what for. The conceptual model of storyboarding (e.g. see [25]) takes this up by providing an integrated model comprising the story space capturing the stories and the plot, actors and tasks. Inspired by approaches in theatre and film, the story space comprises scenes and actions on these scenes, and the plot describes the details of the action scheme. Furthermore, the model describes actors in these scenes, i.e. groups of users, which leads to roles, profiles, goals, preferences, obligations and rights. The actors are linked to the story space by the means of tasks.

5.1 The Story Space in the Codesign Approach

The *story space* consists of a well-integrated set of stories and can be modelled by many-dimensional (multilayered) graphs. A *story* is a run through the story space by a collaboration of users. Typically, each user wants something different; the playout environments can be extremely different; the cultures are distinct; users require a flexible and dynamic adaptation of the system; and they require information and features in a varying granularity. Users may be grouped by the tasks they perform, their profile and their life cases. Such groups are called *actors*.

A story is composed of scenes. Each scene belongs to a general activity. Scenes combine actions of actors into a *plot*. A scene supports enabled actors. Its usage in a

story is controlled by preconditions for entering the scene, by accepting conditions for leaving the scene and by events that opens the scene. Basic dialogue scenes may be combined into a complex dialogue scene based on algebraic operations \square (choice), \parallel (parallel execution), $;$ (sequential execution) and $(.)^*$ (iteration). We may derive extended operations such as simple iteration $(.)^+$ and optional execution $\square skip$. We represent basic dialogue scenes with ellipses. The transitions among dialogue scenes are represented by curves. The story space thus consists of dialogue scenes, their control, actors either involved in the story space or enabled for specific scenes, contexts, tasks and transitions among scenes.

Figure 9 depicts the general structure of story spaces. Codesign of information systems relates the story space and the scenes to the data and the functions provided by the database system. Additionally, a user produces data that are transferred to the database system. These data and functions are given on the basis of media objects [24]. The service interface is typically following specific representation styles. These styles allow to derive the presentation layout and playout (see Fig. 2) in the screenography approach [22]. The layout is based on principles of visual communication, cognition and design.

The *story space* combines many different stories. The concrete deployment of a Web information system by users is a view on this story space, i.e. a *scenario* or path in this space for the given user by abstracting from those scenes for which the user is enabled. Story spaces might be complex for e-business, infotainment (also called information site), edutainment (also called e-learning site), identity and collaboration (also called community) Web information systems.

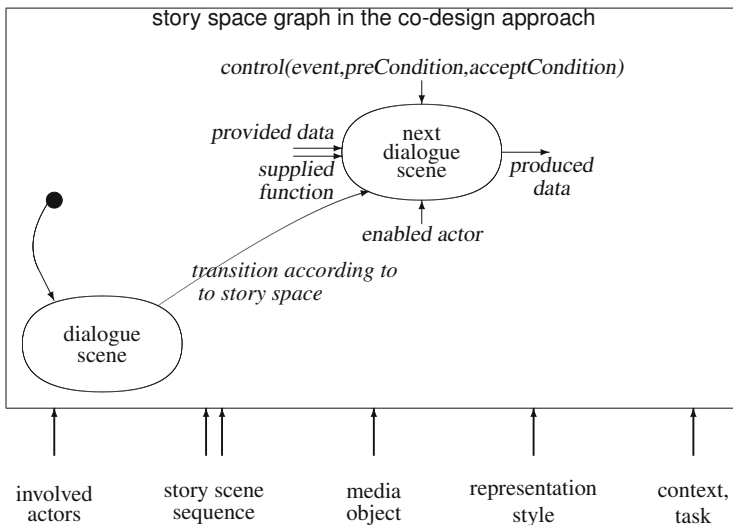


Fig. 9 Codesign of stories (through dialogue scene, their control, involved actors, context, tasks and transitions among scenes), services (information and features), media objects (provided and produced data and functions) and guideline for the presentation system

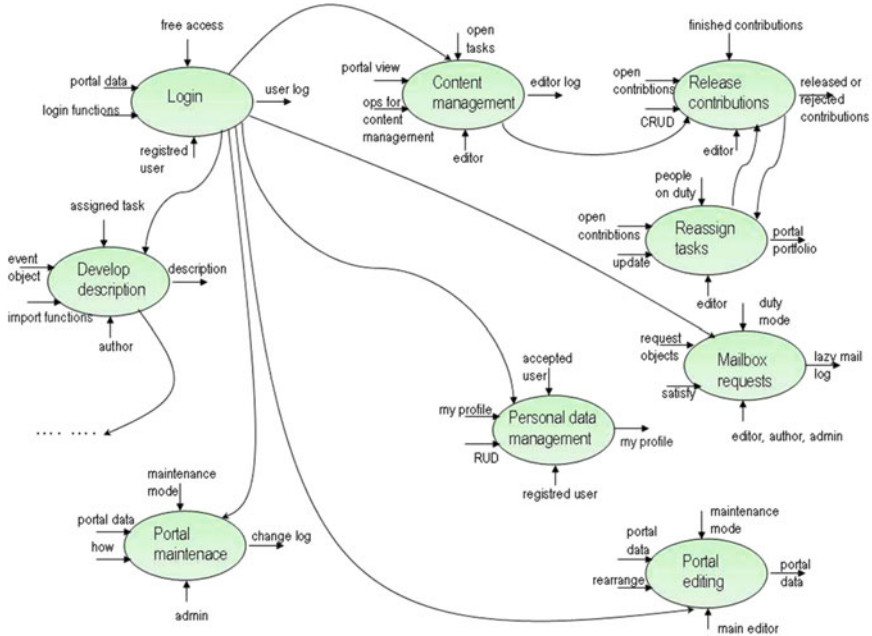


Fig. 10 Cutout of the story space for infotainment sites

For instance, the story space in infotainment contains the editing story. Figure 10 combines the stories of the editor, the author of some data, the administrator of the website and others. Depending on the role that a user plays, the scenario includes scenes such as content management, release of contributions or mailbox requests. The workflow in Fig. 4 is the realisation of such user activities, e.g. in the role of an editor who handles the release of contributions.

5.2 Media Types and Object for Information-Intensive Systems

The classical service-oriented approach in Fig. 6 is based on a mediator between the user and the technical side. Services satisfy the information and activity demand of users by linking information and features to content and functions. However, if we consider information-intensive services, then we extend the technical system with media types. In this case, the technical system does not require a mediating system. Instead, a user calls media objects. The architecture displayed in Fig. 11 neatly supports e-business, edutainment, infotainment, community and identity Web information systems based on classical Web technologies.

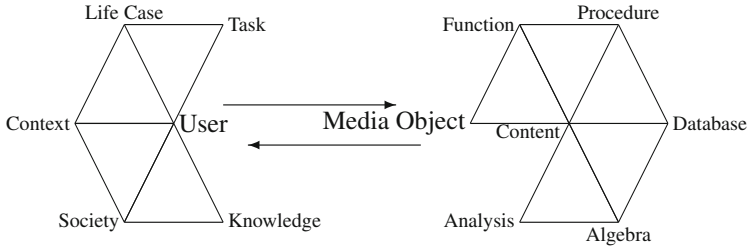


Fig. 11 Media objects for information-intensive systems

5.3 The Onion Approach to Website Realisation

The onion approach [35] to website layout and playout allows the generation of website functions and views for a Web information system. On the outer layer, the presentation facilities may be introduced. Typical functions are style and context functions. Containers that contain the media objects are mapped to the information and the features that users request. Views may be materialised. If they are materialised, then the view handler provides an automatic refreshment support. Thus, we can use the onion system architecture displayed in Fig. 12. This layering is nicely supported by XML infrastructures. The database system view data and functions are represented by an XML document suite. The media object onion is generated by eXtensible Stylesheet Language (XSLT) rules. Such rules can easily be developed for the containers and finally for the scenes in a scenario. Current database management systems such as DB2 neatly support this approach. XML objects are specified by Document Type Definition (DTD) and Dynamic Access Control (DAC) in DB2. We thus implemented the onion approach based on XML transformations in a form depicted in Fig. 13.

This transformation approach has already been used in the DaMiT project [3] and implements an XML suite on top of the relational DBMS DB2. The extended ER model [32] provides a better approach to XML suite generation than relational models or the classical ER model for a number of reasons:

- Structures can be defined already in complex nested formats.
- Types of higher order are supported.
- The model uses cardinality constraints with participation semantics.

We observe further that well-structured XML may be considered to be a restricted form of HERM:

- XML schema and XForms are suited for defining hierarchical extracts of HERM.
- HERM specialisation is based on type specialisation.
- Unary cardinality constraints are supported. If more complex constraints are required, we may use vertical decomposition approaches.
- Variants of Web objects may be referenced by an annotated XHTML Document Navigation Language (XDNL) approach.

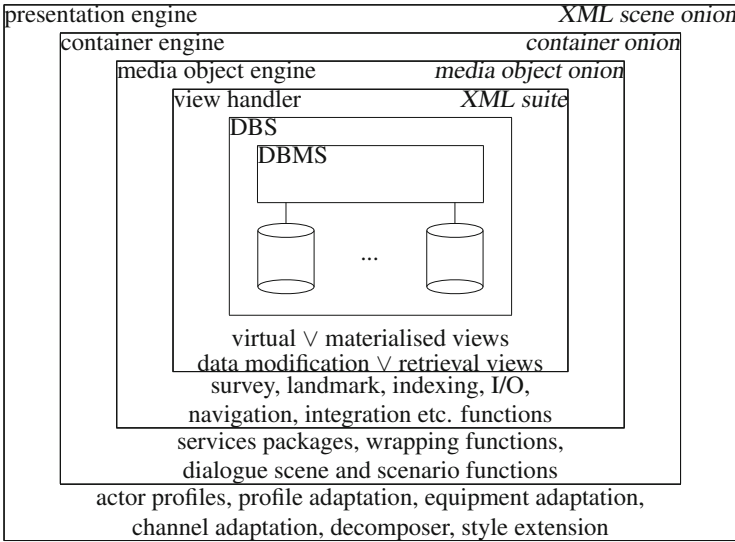


Fig. 12 The onion approach to stepwise generation of XML-based sites

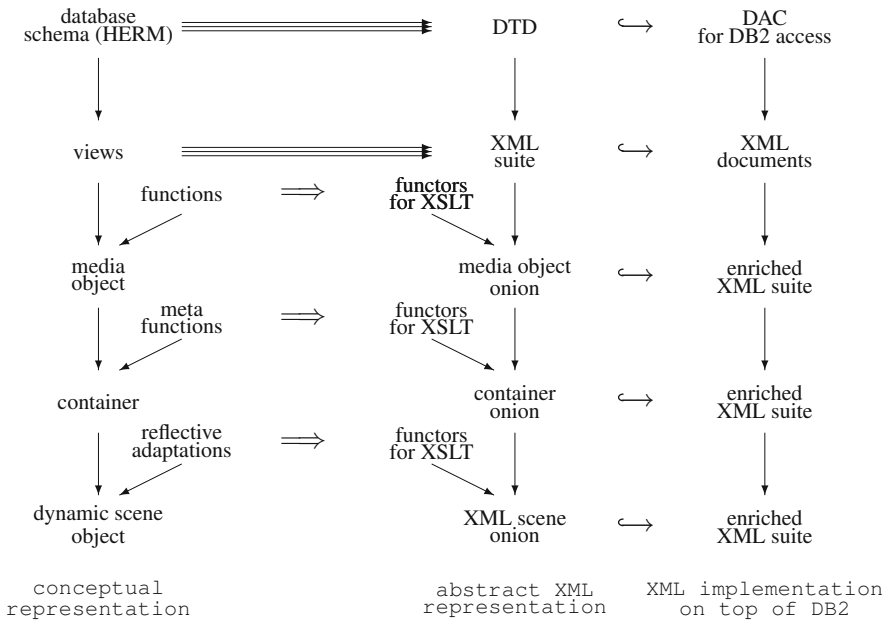


Fig. 13 The general procedure for translation from SiteLang to XML

Therefore, suites of restricted XML documents may be understood as *object-oriented hierarchical database*. If documents are reused by other documents, we associate them via XDNL variants.

Translation of HERM schemata to DTD can be based on a number of approaches which are similar to the translation approaches used for transformation of schemata to hierarchical database schemata:

Full type separation: each entity, relationship and cluster type is represented by their own `<!ELEMENT . . .>` representation. All entity types have an ID which is used through IDREF by other types.

Small star schema representation: the central type of a star schema is represented by its own `<!ELEMENT . . .>` representation which uses components for association to other central types. Star associations to other types are represented through attribute lists and the IDREF #REQUIRED data type.

We observe that the translation is not semantics preserving. All referential constraints must be maintained through application programs.

5.4 Transformation of Web Information Systems

The codesign approach is based on the higher-order entity-relationship model. However, most Web information systems use XML infrastructures for the layout and playout. Therefore, the transformation of database schemata is enhanced with additional elements of the storyboard and by the distribution specification. XML transformation [15, 32] generalises the transformation of ER schemata to hierarchical database schemata. Additionally, the transformation is based on a selection of XML tactics [16], e.g. salami slice or Russian doll schema kinds. Furthermore, the IDREF attribute value allows the introduction of the concept of the shadow type similar to the Conference on Data Systems Languages (CODASYL) approach. There is no need to delve into detail here as they are already well known. There are, however, some specific elements of the transformation process which need a more detailed description.

Our approach to transformation generalises compiler techniques. A typical configuration of a compiler uses *directives*. These directives can directly be mapped to the supporting hardware and software. In the case of Web information systems, such directives are pattern for content organisation pattern, for content presentation, for navigation, for exploration and for search and retrieval.

5.4.1 Deriving the Content Organisation Schema

Content must be provided in a form that each user is able to browse or to zap through, depending on the information and activity demand. The user profile also contains rules concerning the preferred order of the given user, what content

presentation is the most appropriate for what personality and what data have already been given during the scenario that the user has already completed. Therefore, we add to the content schema ordering decisions depending on ordering criteria, on classification schemata that are available and on ordering imposed by the story space.

5.4.2 Deriving the Content Presentation Structure

Content presentation is often organised in a linear and sequential form following the reading style of the user community that is under consideration. Users have their personality profile. It allows derivation of the preferences for reading data, e.g. texts. The Web supports other presentation structures, such as hierarchical or hypermedia structures. Hierarchical structures are easy to develop and difficult to use whenever data sets become larger. Hypermedia structures are highly flexible. There is, however, a risk of lost orientation. Therefore, they require a more sophisticated browsing, zapping and navigating support.

Content presentation must take into consideration parsimony and other quality of use characteristics. It is a challenging task for a user to scroll through a Web page and to remember all of those data which are out of the screen due to scrolling. Therefore, we need to apply techniques that allow the user to see data together with their associated data. The best metaphor for infotainment content presentation seems to be the concertina-type cover.

5.4.3 Deriving the Navigation Structure

A website must support users in the space of the website. The simplest technique for orientation in a website is navigation. We distinguish hierarchical navigation based on the tree paradigm, global navigation within a Web space based on vertical or horizontal walks in this space, local navigation that extends global navigation in a subspace of the Web space and ad hoc navigation via meaningful anchors and hyperlinks. Navigation aids are explicit maps throughout the Web space, indexing and cataloguing schemata, headers and teasers, adaptable guided tours and meaningful anchors and icons.

5.4.4 Extending Navigation by Exploration Techniques

Users may be supported for navigation by corresponding exploration techniques of the Web space. Typical exploration techniques are fish-eye viewing techniques (3D-fish-eye, adorned fish-eye, filtered fish-eye), transformation techniques [radial (locally or globally), orthogonal (locally or globally), three-dimensional (implicitly or explicitly)], zoom techniques, nonlinear techniques (e.g. with the focus point or with multipoint hyperbolic planes) and adoption techniques based on the relevance or importance of the content.

5.4.5 Extending by Retrieval and Search Features

Information-intensive WIS must be supported by sophisticated retrieval and search feature for all the seven kinds of search (querying by queries; seeking for data by browsing, understanding and refining; property-based questioning; ferreting out data necessary by discovering; searching by associations and drilling down; casting about and digging into the data; zapping through data sets based on search techniques, e.g. uninformed search). If the size of the data is rather small, then classical querying approaches do not fit. In this case, retrieval is better provided by exploration techniques. If the data in the databases are not frequently changed and the Web information system is traffic intensive, then all potential retrieval features may be supplied by preprepared materialised or well-indexed data marts.

Additionally, the user language may not match with the interpretation of the WIS, e.g. for quantifiers, connectives and open-world questions. Therefore, the semantic space of the user should be mapped to the formal semantic space of the WIS. Furthermore, users need feedback and metadata on the quality of the delivered data. Finally, the presentation of search results can be based on text reading approaches (e.g. Google) or on mind map techniques (e.g. KartOO).

5.5 Mapping of the Website Specification to Business Layer Models

Business use cases [21] are constructed for business processes as bird’s-eye views of desired business and system behaviour. They correspond to system features. Business use cases are turned into use cases during requirements analysis. Actors are external, i.e. passive with regard to use cases. However, use cases reflect concerns of system construction, e.g. inheritance of functionality and data (“extends”) or composition (“uses”) (Fig. 14).

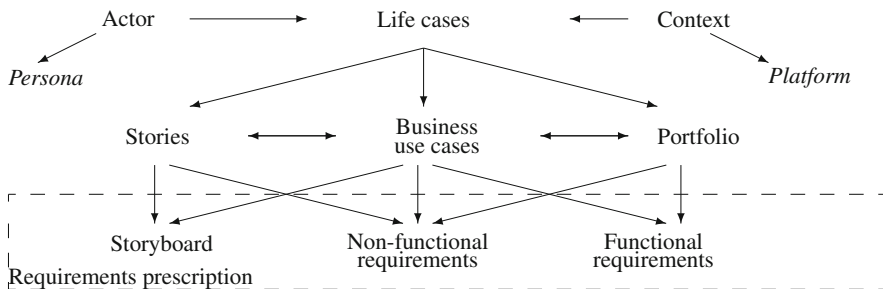


Fig. 14 The use of application domain information for requirements elicitation and analysis

We embed our specification approach into classical software engineering approaches that use application domain description, requirements prescription and software specification as three different concerns in the system development triptych [4]. The starting point in our codesign framework is the users or a group of users with a similar profile and task portfolio (called actors), their life cases and their context, e.g. the platform to be used. Life cases are the basis for a number of stories, a set of business use cases and a portfolio that supports all tasks of users.

Classical requirements prescription is distinguished between functional and nonfunctional requirements. These requirements do not capture the flow of actions that a user performs. Later, business rules and business workflows are added. In most cases, they are unrelated to the user stories. They are developed on their own. In our approach, however, we use stories that are already developed and combine these stories into a storyboard. This storyboard can be associated with workflows. A scenario can be supported by several workflows and a workflow can support several scenes of a scenario.

5.6 Web Page Extraction

Our approach also allows direct extraction of the Web page which supports the basic scenes. Let us consider the login scene in Fig. 10. Users may play the role of the actors, editor, author, main editor or administrator.

A Web page carries four aspects: the actors, the topic of the page, the media objects and the restrictions, especially the time restrictions. These four aspects are interrelated. Actors are supported in their work by the media objects both for private use (e.g. workspace, unfinished contributions and their personal workspace) and collaboration use (e.g. shared contributions). One specific media object is the personal working room media object. It generalises the concept of a session. The work of an actor is governed by the intention and the tasks which must be completed. We combine intention and tasks into the topic of the page. The topic is related to the constraints and restrictions, e.g. time restrictions for the completion of a contribution depending on deadlines and phases within the editing activities. The login Web page is supported by media objects, e.g. editing media objects. Figure 15 shows a diamond representation of the four aspects with their associations.

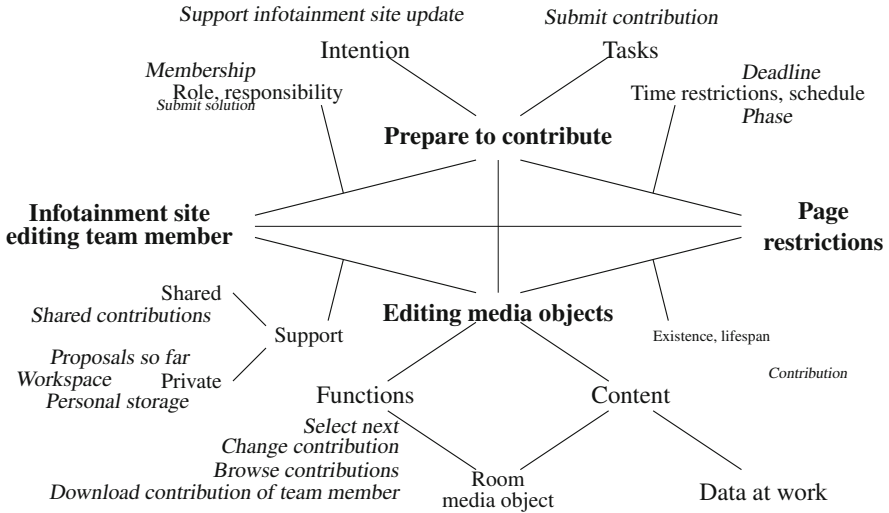


Fig. 15 Aspects to be supported by a Web page during editing in infotainment

5.7 Configuration of the Web Page to User Context by Containers

Users, e.g. visitors of a website, may use a large variety of media ranging from small display media such as WAP displays to middle display media such as smartphone interfaces to full screens. We thus need a way to deliver the content and the functions to the user in such a form that the layout is adapted to the current interface.

Most websites that adapt to the user develop different pages for each kind of screen that they support. Since this kind of duplication is laborious, time consuming and error sensitive, websites have lately been limited to specific browsers and interfaces. However, the theory of media types allows the use of a different approach. Each scene is supported by a number of media objects. These media objects are innerly structured. Therefore, they can be decomposed into several views on top of these objects in dependence on the interface context.

Containers [7] are media objects that support content shipping, parametrisation of presentation and context adaptation. They are used for transfer of information to users according to their current demand and the corresponding dialogue scene. Size and presentation of containers depend on the restrictions of the user’s environment. Containers are obtained by adding functions for adaptation and unloading. Scene media objects are specialisations of containers by adaption of the container to the user (profile), the environment and the history. The adaptation of the media object to the current context is based on measure rules which support translation of different

scales, ordering rules for ordering of components in the container, adhesion rules that maintain coherence of components in a media object and hierarchy meta-rules similar to OLAP functions.

Let us consider the visit of a user interested in sporting events. This activity is supported by the media type in Fig. 5. The adhesion of *clubs* to *events* is higher than the one of *locations* and *time* to *event*. Several hierarchies exist such as the time hierarchy (year, month, week, day, daytime) and the location hierarchy (region, town, village, quarter, street), the latter one being fanned. Therefore, the media object has the natural internal land map shown in Fig. 16. Containers may be extended by docketets which allow tracking of the current usage of the content. They contain data on the content; on the delivery instruction; on the parameters

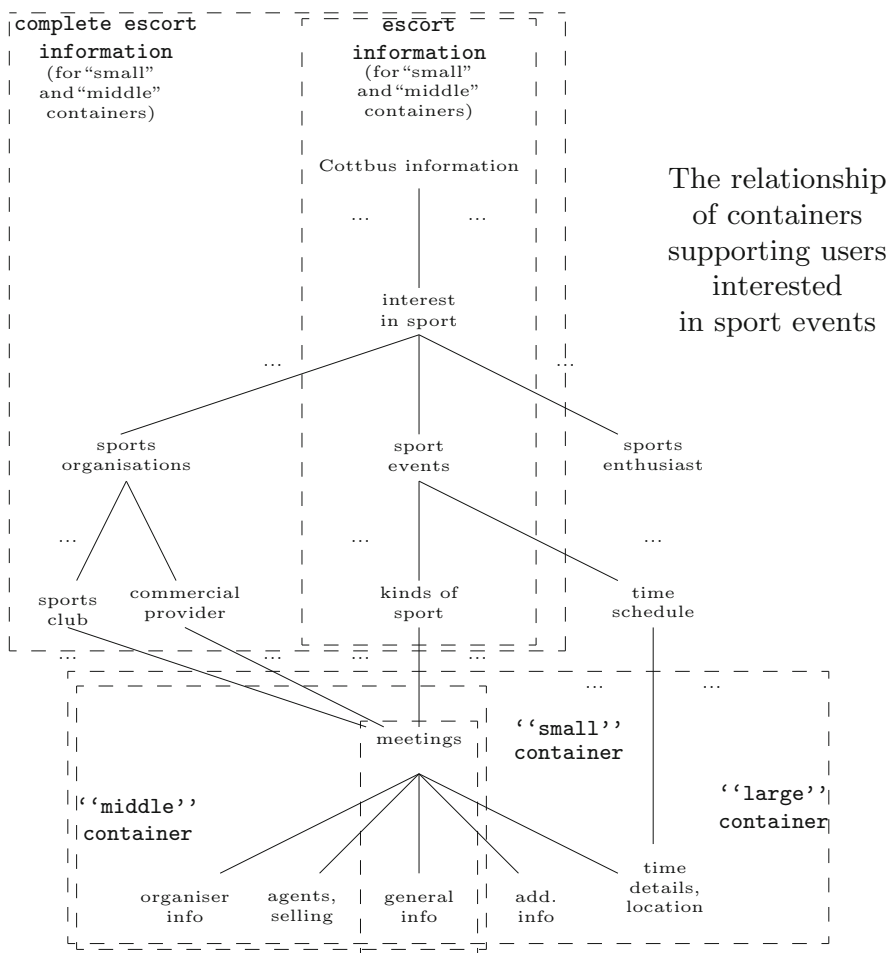


Fig. 16 The orientation and information containers satisfying interest in sport events

of functions for opening the document; on associations to other media objects; on metadata such as resources, restriction, copyright, roles, distribution policy, etc.; on the content providers, content reviewers and review evaluators with quality control policies; on applicable workflows and the current status of completion; and on the receipt of the document which enable in tracing the media object life cycle. The overlay structuring of dockets for containers is supported by the onion approach indicated in Fig. 12.

Media types support a variety of functions such as generalisation, specialisation, reordering, browsing, sequentialisation, linking, survey, searching and join functions. These functions are also provided by the container.

In Fig. 16, three different kinds of containers are used: small containers for the most essential and not decomposable components, middle containers with more elaborated data and full containers for the complete information. We can also distinguish an orientation container from the information container. The information container reuses data from the orientation container as escort information.

In the event example, the order is either specified by the scenario in the story space or by the order of information presentation, e.g. it is assumed that information on actual sporting events is shown before information on previous sporting events is given.

6 Conclusion

6.1 *From Database Development to Codesign of Web Information Systems*

Database development has mainly been considered as development of database structuring. Functionality and interactivity specification has been neglected in the past. The derivability of functionality has been a reason for this restriction of the database design approach. At the same time, applications and the required functionality have become more complex. Functionality specification may be based on workflow engines. Interaction support is often not specified but hidden within the interfaces. It may, however, be specified through the story space and the supporting media-type suite. Distributed applications are based on an explicit specification of import/export views, on services provided and on exchange frames. The codesign approach aims in bridging all these different aspects of applications and, additionally, provides a sound methodology for development of all these aspects.

6.2 Applications and Assessment of the Codesign Methodology

The codesign methodology has been practically applied in a large number of information system projects and has nevertheless a sound theoretical basis. We do not want to compete with UML, but we do support system development on a sound basis without ambiguity, ellipses and conceptual mismatches.

Methodologies must conform both with the SPICE v. 2.0 and SW-CMM v. 2.0 requirements for consistent system development. The codesign framework is based on a stepwise refinement along the abstraction layers. Since the four aspects of information systems—structuring, functionality, distribution and interactivity—are interrelated, they cannot be developed separately. The presented codesign framework is underpinned by a methodology for stepwise development. It has been assessed and is testified [8] to be on level 3 in the SPICE framework.

References

1. Amarakoon, S., Dahanayake, A., Thalheim, B.: A framework for modelling medical diagnosis and decision support services. *Int. J. Digit. Inf. Wirel. Commun.* **2**(4), 7–26 (2012)
2. Beeri, C., Thalheim, B.: Identification as a primitive of database models. In: *Proceedings of FoMLaDO'98*, pp. 19–36. Kluwer, London (1999)
3. Binemann-Zdanowicz, A., Kaschek, R., Kuss, T., Schewe, K.-D., Thalheim, B., Tschiedel, B.: A conceptual view of electronic learning systems. *Educ. Inf. Technol.* **10**, 83–110 (2005)
4. Bjørner, D.: *Software Engineering 3: Domains, Requirements, and Software Design*. Springer, Berlin (2006)
5. Chen, P.P.: The entity-relationship model: toward a unified view of data. *ACM Trans. Database Syst.* **1**(1), 9–36 (1976)
6. Dahanayake, A., Thalheim, B.: A conceptual model for IT service systems. *J. Univers. Comput. Sci.* **18**(17), 2452–2473 (2012)
7. Feyer, T., Schewe, K.-D., Thalheim, B.: Conceptual design and development of information services. In: *ER'98*, pp. 7–20 (1998)
8. Fiedler, G., Jaakkola, H., Mäkinen, T., Thalheim, B., Varkoi, T.: Application domain engineering for web information systems supported by SPICE. In: *Proceedings of SPICE'07*. IOS Press, Bangkok (2007)
9. Gogolla, M.: *An Extended Entity-Relationship Model - Fundamentals and Pragmatics*. Lecture Notes in Computer Science, vol. 767. Springer, Berlin (1994)
10. Hartmann, S.: Reasoning about participation constraints and Chen's constraints. In: *ADC. CRPIT*, vol. 17, pp. 105–113. Australian Computer Society, Sydney (2003)
11. Hartmann, S., Hoffmann, A., Link, S., Schewe, K.-D.: Axiomatizing functional dependencies in the higher-order entity-relationship model. *Inf. Process. Lett.* **87**(3), 133–137 (2003)
12. Heinrich, L.J.: *Informationsmanagement: Planung, Überwachung und Steuerung der Informationsinfrastruktur*. Oldenbourg Verlag, München (1996)
13. Heinrich, L.J., Heinzl, A., Riedl, R.: *Wirtschaftsinformatik: Einführung und Grundlegung*, 4th edn. Springer, Berlin (2011)
14. Hohenstein, U.: *Formale Semantik eines erweiterten Entity-Relationship-Modells*. Teubner, Stuttgart (1993)
15. Kleiner, C., Lipeck, U.W.: Automatic generation of XML DTDs from conceptual database schemas. In: *GI Jahrestagung* (1), pp. 396–405 (2001)

16. Klettke, M.: Modellierung, Bewertung und Evolution von XML-Dokumentkolektionen. Advanced Ph.D. (Habilitation Thesis), Rostock University (2007)
17. König, H.: Protocol Engineering: Prinzip, Beschreibung und Entwicklung von Kommunikationsprotokollen. Teubner, Stuttgart (2003)
18. Levene, M., Loizou, G.: A Guided Tour of Relational Databases and Beyond. Springer, Berlin (1999)
19. Liddle, S.W., Embley, D.W., Woodfield, S.N.: Cardinality constraints in semantic data models. *Data Knowl. Eng.* **11**, 235–270 (1993)
20. Lockemann, P.C.: Information system architectures: from art to science. In: Proceedings of BTW'2003, pp. 1–27. Springer, Berlin (2003)
21. Maciaszek, L.: Requirements Analysis and Design. Addison-Wesley, Harlow (2001)
22. Moritz, T., Noack, R., Schewe, K.-D., Thalheim, B.: Intention-driven screenography. In: ISTA 2007. *Lecture Notes in Informatics*, vol. 107, pp. 128–139 (2007)
23. Schewe, K.-D.: The specification of data-intensive application systems. Advanced Ph.D. (Habilitation Thesis), Brandenburg University of Technology at Cottbus (1994)
24. Schewe, K.-D., Thalheim, B.: Structural media types in the development of data-intensive web information systems. In: *Web Information Systems*, pp. 34–70. IDEA Group, Hershey (2004)
25. Schewe, K.-D., Thalheim, B.: Conceptual modelling of web information systems. *Data Knowl. Eng.* **54**, 147–188 (2005)
26. Schewe, K.-D., Thalheim, B.: Usage-based storyboarding for web information systems. Technical Report 2006-13, Christian Albrechts University Kiel, Institute of Computer Science and Applied Mathematics, Kiel (2006)
27. Schewe, K.-D., Thalheim, B.: Development of collaboration frameworks for web information systems. In: 20th International Joint Conference on Artificial Intelligence, Section EMC07 (Evolutionary models of collaboration), Hyderabad, pp. 27–32 (2007)
28. Schewe, K.-D., Thalheim, B.: Life cases: an approach to address pragmatics in the design of web information systems. In: Filipe, J., Cordeiro, J., Encarnacao, B., Pedrosa, V. (eds.) *Proceedings of WebIST*, vol. II (WIA), pp. 5–12 (2007)
29. Schewe, K.-D., Thalheim, B.: Life cases: a kernel element for web information systems engineering. In: *Web Information Systems and Technologies. Lecture Notes in Business Information Processing*, vol. 8, pp. 139–156. Springer, Heidelberg (2008)
30. Srinivasa, S.: An algebra of fixpoints for characterizing interactive behavior of information systems. Ph.D. thesis, BTU Cottbus (2000)
31. Thalheim, B.: Dependencies in Relational Databases. Teubner, Leipzig (1991)
32. Thalheim, B.: Entity-Relationship Modeling Foundations of Database Technology. Springer, Berlin (2000)
33. Thalheim, B.: Codesign of structuring, functionality, distribution and interactivity. *Aust. Comput. Sci. Commun.* **31**(6), 3–12 (2004). *Proc. APCCM'2004*
34. Thalheim, B.: The enhanced entity-relationship model. In: *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 12, pp. 165–208. Springer, Berlin (2011)
35. Thalheim, B., Düsterhöft, A.: Sitelang: conceptual modeling of internet sites. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) *ER. Lecture Notes in Computer Science*, vol. 2224, pp. 179–192. Springer, Berlin (2001)
36. Weske, M.: Business Process Management: Concept, Language, Architecture. Springer, Heidelberg (2007)
37. Whorf, B.L.: Lost Generation Theories of Mind, Language, and Religion. Popular Culture Association. University Microfilms International, Ann Arbor (1980)