Communications in Computer and Information Science 336

Editorial Board

Simone Diniz Junqueira Barbosa
Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil

Phoebe Chen
La Trobe University, Melbourne, Australia

Alfredo Cuzzocrea
ICAR-CNR and University of Calabria, Italy

Xiaoyong Du
Renmin University of China, Beijing, China

Joaquim Filipe
Polytechnic Institute of Setúbal, Portugal

Orhun Kara
TÜBİTAK BİLGEM and Middle East Technical University, Turkey

Tai-hoon Kim
Konkuk University, Chung-ju, Chungbuk, Korea

Igor Kotenko
St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences, Russia

Dominik Ślęzak
University of Warsaw and Infobright, Poland

Xiaokang Yang
Shanghai Jiao Tong University, China

Reiner Hähnle Jens Knoop
Tiziana Margaria Dietmar Schreiner
Bernhard Steffen (Eds.)

Leveraging Applications of Formal Methods, Verification, and Validation

International Workshops
SARS 2011 and MLSC 2011
Held Under the Auspices of ISoLA 2011
in Vienna, Austria, October 17-18, 2011
Revised Selected Papers

Springer

Volume Editors

Reiner Hähnle
TU Darmstadt, Department of Computer Science
64289 Darmstadt, Germany
E-mail: haehnle@cs.tu-darmstadt.de

Jens Knoop
TU Vienna, Faculty of Informatics
1040 Vienna, Austria
E-mail: knoop@complang.tuwien.ac.at

Tiziana Margaria
University Potsdam, Institute of Informatics
14482 Potsdam, Germany
E-mail: margaria@cs.uni-potsdam.de

Dietmar Schreiner
TU Vienna, Faculty of Informatics
1040 Vienna, Austria
E-mail: schreiner@complang.tuwien.ac.at

Bernhard Steffen
TU Dortmund, Faculty of Informatics
44227 Dortmund, Germany
E-mail: steffen@cs.tu-dortmund.de

ISSN 1865-0929
e-ISSN 1865-0937
ISBN 978-3-642-34780-1
e-ISBN 978-3-642-34781-8
DOI 10.1007/978-3-642-34781-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012951161

CR Subject Classification (1998): I.2.9, I.2.6, I.2.0-1, H.2.8, I.2.4, D.2.11, D.2.4-5, D.2.1, J.2, F.1.1, H.3.5, I.5.3-4

© Springer-Verlag Berlin Heidelberg 2012
This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

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

Printed on acid-free paper

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

Preface

This issue contains a selection of revised papers that were presented at the Software Aspects of Robotic Systems (SARS 2011) Workshop and the Machine Learning for System Construction (MLSC 2011) Workshop held during October 17–18 in Vienna, Austria, under the auspices of the International Symposium Series on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA).

Both workshops are in line with the general mission statement of the ISoLA Symposium series. That is to provide a forum for developers, users, and researchers for discussing issues related to the adoption and the use of rigorous tools for specification, analysis, verification, certification, construction, test, and maintenance of systems from their domain-specific point of view. Thereby, the ISoLA symposia contribute to bridging the gap between designers and developers of (formal methods based) rigorous tools, and users in engineering and in other disciplines.

The SARS workshop and the MLSC workshop pursue this mission within the domains of software aspects of robotic systems and machine learning for system construction.

The timeliness of the SARS workshop stems from the fact that development of autonomous robotic systems experienced a remarkable boost within the last few years. Away from stationary manufacturing units, current robots have grown up into autonomous, mobile systems that not only interact with real-world environments, but also fulfill mission critical tasks in collaboration with human individuals on a reliable basis. Typical fields of application are unmanned vehicles for exploration but also for transportation, reconnaissance and search-and-rescue in hazardous environments, and ambient-assisted living for elderly or disabled people.

Hence, algorithms in cognition, computer vision, and locomotion have become hot-spots of research and development. In addition, modern concepts like evolutionary and bio-inspired design have entered the stage to tackle open issues in robotics and to cope with domain-specific properties such as inherent indeterminism.

The back-side of this boost is an even larger increase in complexity of modern robotic systems. Numerous actuators and sensors have to be controlled simultaneously. Complex actions have to be performed via timed parallel execution of multiple instruction streams on distinct electronic control units. Autonomy, especially long-term autonomy as required by deep-sea or space exploration missions, necessitates features of fault-tolerance, error recovery, or at least well-defined fallbacks. Owing to the physical interaction of robots with the real world, safety violations are extremely harmful, in the worst case they might lead to severe damage and even to casualties.

The timeliness of the MLCS workshop follows from the fact that even state-of-the-art systems often lack adequate specifications or make use of un/under-specified components. In fact, the popular component-based software design paradigm naturally leads to under-specified systems, as most libraries only provide very partial specifications of their components. Moreover, revisions and last-minute changes typically hardly enter the system specification.

As observable in many practical contexts, revision cycles are often extremely short, making the maintenance of specifications unrealistic, and at the same time necessitating extensive testing effort. More generally, the lack of documentation is sadly perceived in many places, among which quality control is one of the most prominent.

Machine learning has been proposed to overcome this situation by automatically "mining" and then updating the required information. Promising results have been obtained here using active automata-learning technology, and there seems to be a high potential to also exploit other maching-learning techniques.

Both the SARS workshop and the MLSC workshop attracted researchers and practioners from academia and industry and provided a lively forum for them to present and discuss their most recent research results in the respective fields of the two workshops.

The present issue of *Communications in Computer and Information Science* contains the revised versions of selected papers that were presented at the workshops. These papers have undergone a second round of reviewing, and reflect the suggestions of the reviewers as well as feedback from the presentation and discussion of the papers at the workshops.

The topics covered by the papers of the SARS and the MLSC workshop demonstrate the breadth and the richness of the respective fields of the two workshops stretching from robot programming to languages and compilation techniques, to real-time and fault tolerance, to dependability, software architectures, computer vision, cognitive robotics, multi-robot coordination, and simulation to bio-inspired algorithms, and from machine learning for anomaly detection, to model construction in software product lines to classification of Web service interfaces.

In addition the SARS workshop hosted a special session on the recently launched KOROS project on *collaborating robot systems* that is borne by a consortium of researchers of the faculties of architecture and planning, computer science, electrical engineering and information technology, and mechanical and industrial engineering at the *Vienna University of Technology*. The four papers devoted to this session highlighted important research directions pursued in this interdisciplinary research project.

Finally, we would like to thank the many individuals who contributed to making the ISoLA 2011 worshops a success. First of all, we thank the members of the SARS and MLSC Program Committees for their dedicated and diligent work of selecting the papers for presentation at the two workshops. We also thank the authors who submitted a paper to one of the workshops. Our special thanks go to the invited keynote speakers at the SARS workshop, Davide Brugali

(Università degli Studi di Bergamo), Rick Middleton (National University of Ireland Maynooth), Daniele Nardi (Sapienza Università di Roma), and Trevor Taylor (Microsoft, Redmond). Last but not least, we thank Alfred Hofmann, Anna Kramer, and Leonie Kunz at Springer for publishing these proceedings in the CCIS series and for the smooth co-operation.

August 2012

Reiner Hähnle
Jens Knoop
Tiziana Margaria
Dietmar Schreiner
Bernhard Steffen

ISoLA 2011 Workshop Organization

General Chair

Tiziana Margaria	University of Potsdam, Germany

SARS 2011 Workshop Chairs

Jens Knoop	TU Vienna, Austria
Dietmar Schreiner	TU Vienna, Austria

Program Committee

José María Cañas Plaza	Universidad Rey Juan Carlos, Fuenlabrada, Spain
Markus Bader	TU Vienna, Austria
Karl M. Göschka	TU Vienna, Austria
Vincent Hugel	Université de Versailles, France
Jens Knoop, Co-chair	TU Vienna, Austria
Gerald Steinbauer	TU Graz, Austria
Dietmar Schreiner, Co-chair	TU Vienna, Austria
Stefan Tasse	TU Dortmund, Germany
Markus Vincze	TU Vienna, Austria
Arnoud Visser	Universiteid van Amsterdam, The Netherlands
Franz Wotawa	TU Graz, Austria

MLCS 2011 Workshop Chairs

Reiner Hähnle	TU Darmstadt, Germany
Bernhard Steffen	TU Dortmund, Germany

Program Committee

Michael Felderer	Universität Innsbruck, Austria
Falk Howar	TU Dortmund, Germany
Reiner Hähnle, Co-chair	TU Darmstadt, Germany
Valérie Issarny	INRIA, Paris-Rocquencourt, France
Richard Johansson	University of Gothenburg, Sweden

Fabio Massaci	Università di Trento, Italy
Alessandro Moschitti	Università di Trento, Italy
Tomas Piatrik	Queen Mary University of London, UK
Riccardo Scandariato	KU Leuven, Belgium
Ina Schäfer	TU Braunschweig, Germany
Bernhard Steffen, Co-chair	TU Dortmund, Germany

Table of Contents

Software Aspects of Robotic Systems (SARS 2011)

A Role-Based Language for Collaborative Robot Applications 1
Sebastian Götz, Max Leuthäuser, Jan Reimann, Julia Schroeter, Christian Wende, Claas Wilke, and Uwe Aßmann

Efficient Localization for Robot Soccer Using Pattern Matching 16
Thomas Whelan, Sonja Stüdli, John McDonald, and Richard H. Middleton

A NUPlatform for Software on Articulated Mobile Robots 31
Jason Kulk and James S. Welsh

Service Component Architectures in Robotics: The SCA-Orocos Integration 46
Davide Brugali, Luca Gherardi, Markus Klotzbücher, and Herman Bruyninckx

Safe Autonomous Transport Vehicles in Heterogeneous Outdoor Environments 61
Tobe Toben, Sönke Eilers, Christian Kuka, Sören Schweigert, Hannes Winkelmann, and Stefan Ruehrup

Adaptive Autonomous Systems – From the System's Architecture to Testing 76
Franz Wotawa

Representing Knowledge in Robotic Systems with KnowLang 91
Emil Vassev and Mike Hinchey

Object Detection and Classification for Domestic Robots 106
Markus Vincze, Walter Wohlkinger, Sven Olufs, Peter Einramhof, Robert Schwarz, and Karthik Varadarajan

A Software Integration Framework for Cognitive Systems 121
Michael Zillich, Wolfgang Ponweiser, and Markus Vincze

Special Session on KOROS

KOROS Initiative: Automatized Throwing and Catching for Material Transportation 136
Martin Pongratz, Klaus Pollhammer, and Alexander Szep

Cognitive Decision Unit Applied to Autonomous Robots 144
Dietmar Bruckner and Friedrich Gelbard

Building iRIS: A Robotic Immune System 150
Dietmar Schreiner

Towards Reorientation with a Humanoid Robot 156
Dietmar Bruckner, Markus Vincze, and Isabella Hinterleitner

Machine Learning for System Construction (MLSC 2011)

Monitoring Anomalies in IT-Landscapes Using Clustering Techniques and Complex Event Processing 162
Matthias Gander, Michael Felderer, Basel Katt, and Ruth Breu

A Hierarchical Variability Model for Software Product Lines 181
Dilian Gurov, Bjarte M. Østvold, and Ina Schaefer

Learning-Based Software Testing: A Tutorial 200
Karl Meinke, F. Niu, and M. Sindhu

Machine Learning for Automatic Classification of Web Service Interface Descriptions 220
Amel Bennaceur, Valérie Issarny, Richard Johansson, Alessandro Moschitti, Daniel Sykes, and Romina Spalazzese

The Teachers' Crowd: The Impact of Distributed Oracles on Active Automata Learning 232
Falk Howar, Oliver Bauer, Maik Merten, Bernhard Steffen, and Tiziana Margaria

Automata Learning with On-the-Fly Direct Hypothesis Construction . . . 248
Maik Merten, Falk Howar, Bernhard Steffen, and Tiziana Margaria

Author Index 261

A Role-Based Language for Collaborative Robot Applications

Sebastian Götz, Max Leuthäuser, Jan Reimann, Julia Schroeter,
Christian Wende, Claas Wilke, and Uwe Aßmann

Technische Universität Dresden
Institut für Software- und Multimediatechnik
D-01062, Dresden, Germany
sebastian.goetz@acm.org, max.leuthaeuser@googlemail.com,
{jan.reimann,julia.schroeter,c.wende,
claas.wilke,uwe.assmann}@tu-dresden.de

Abstract. The recent progress in robotic hard- and software motivates novel, collaborative robot applications, where multiple robots jointly accomplish complex tasks like surveillance or rescue scenarios. Such applications impose two basic challenges: (1) the complexity of specifying collaborative behavior and (2) the need for a flexible and lightweight communication infrastructure for mobile robot teams. To address these challenges, we introduce NaoText, a role-based domain-specific language for specifying collaborative robot applications. It contributes dedicated abstractions to conveniently structure and implement collaborative behavior and thus, addresses the complexity challenge. To evaluate NaoText specifications, we introduce an interpreter architecture that is based on representational state transfer (REST) as a lightweight and flexible infrastructure for communication among robot teams. We exemplify the application of NaoText using an illustrative example of robots collaborating in a soccer game and discuss benefits and challenges for our approach compared to state-of-the-art in robot programming.

1 Introduction

Recent progress in robotic hard- and software has led to more sophisticated and easier programmable robot platforms. In addition to their classical domains in fabrication and research, robots are expected to become affordable for all-day applications within the coming decade, e.g., in home entertainment or facility management. This leads to new application scenarios where robots are distributed, mobile, and communicating, solving complex tasks in collaborating teams, e.g., rescue or surveillance missions.

Typically, robot behavior is implemented in one of two different ways: (1) general purpose languages (GPLs) such as C, C++, or Java can be used for robot programming or (2) abstract domain-specific languages (DSLs) (e.g., Microsoft's visual programming language (VPL), Choregraphe[1] from Aldebaran, or

[1] http://www.aldebaran-robotics.com/en/Discover-NAO/Software/choregraphe.html

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 1–15, 2012.
© Springer-Verlag Berlin Heidelberg 2012

LabView for Mindstorms NXT[2]) can be used and have gained momentum in the robotics community [8]. However, implementing collaborative scenarios is still a challenging task, as the declarative expression of collaborations as first-class programming constructs is not supported by any of these languages.

For the future of robot programming, we envision a role-based [23] approach to express and control robot collaborations. Achieving this vision induces two main requirements: First, abstraction from robot-platform specific concepts and support for the expression of collaborative robot applications in an easy and comprehensive way. Second, a flexible and lightweight communication infrastructure to enable execution and coordination of collaborative tasks among robot teams. We argue that such coordination can be realized using a (possibly external) controller that executes a given collaboration specification and monitors and steers collaborative robot behavior (cf. Fig. 1).

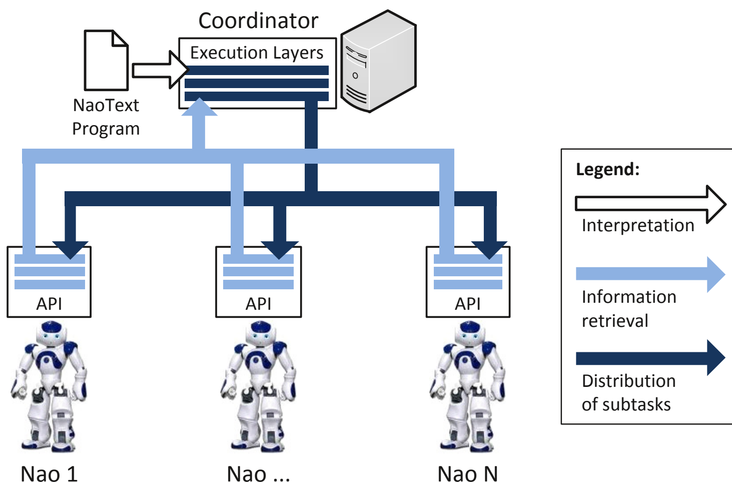


Fig. 1. Naos Working in Collaboration via a central Coordinator

To address these requirements, this paper introduces NaoText, a role-based DSL for programming collaborative robot applications. NaoText is based on the concept of *contexts* allowing convenient specifications of how the entities of a system behave in respective contexts. This partial (i.e., context-dependent) behavior is denoted by *roles*, which interact to achieve collaborative behavior. As depicted in Fig. 1, NaoText is executed by a Java-based interpreter that is running on a central coordinator scheduling tasks of collaborative applications to several robots communicating with the coordinator. The communication is based on a lightweight, representational state transfer (REST)-ful service-oriented architecture (SOA) that consists of and interconnects two stacks of layers. The first stack runs on the coordinator and evaluates a NaoText-based collaboration specification. The second stack of layers runs on each robot and exposes its

[2] http://www.ni.com/academic/mindstorms/

functionality using REST-ful Web services that are accessed and invoked by the coordinator. This paper introduces both NaoText and its underlying implementation architecture. Furthermore, we exemplify the application of NaoText using a soccer game as an exemplary collaborative robot application and discuss it's advantages and disadvantages.

The remainder of the paper is structured as follows. In Sect. 2 we present our exemplary robot soccer application. In Sect. 3, we shortly outline the architecture implemented to evaluate NaoText projects. Sect. 4 introduces the foundations of NaoText for specifying collaborative robot applications by using the introduced example. We discuss benefits of and challenges for our approach in Sect. 5. A detailed discussion of related work follows in Sect. 6. Finally, Sect. 7 concludes this paper and outlines future work.

2 Motivating Example

To illustrate the need for a language which allows to express collaborations between robots, we introduce a motivating example in this section. A well-known testbed for the robotics community and interdisciplinary research is the RoboCup.[3] Robot soccer—an area of competition amongst others at the RoboCup—is a highly dynamic, collaborative, and context-sensitive game. Hence, for our motivating example, we choose a robot soccer scenario with Nao robots. The Nao—developed by Aldebaran Robotics—is a humanoid robot produced in series and offering a standard platform. It has been chosen as the model for the standard platform league at the RoboCup in 2011.

The basic structure of robot soccer is analog to usual soccer. Two teams play against each other and a referee takes care that every participant respects the rules. Each team comprises several roles: goalkeepers, strikers, defensive players, and sweepers. Depending on the number of robots per team,[4] only some of these roles are bound to players during a match. Further, a single role can be played by multiple robots (e.g., the defensive player role) or can be bound to only one robot per team (e.g., the goalkeeper role).

Notably, several contexts can be identified for a soccer match. Besides each team forming a separate context, situations at runtime determine contexts, too. If, for example, two players intend to pass the ball from one to the other, this pass defines a context of its own. Further examples include the shot on goal, the tackling, and the corner kick, to name but a few. Each context defines roles, which in turn specify how the participants in this context behave. For example, a robot, which intends to pass the ball to a teammate, starts playing the `Sender` role in the `Pass` context. It has to compute the angle to the receiving robot (`Receiver`) and shoot the ball in case there are no opponents in the trajectory. Fig. 2 depicts the central concepts used in our example. It first introduces the context `Football_Game` defining the above mentioned roles of a football team.

[3] http://www.robocup.org/

[4] In robot soccer, less than eleven participants per team are common.

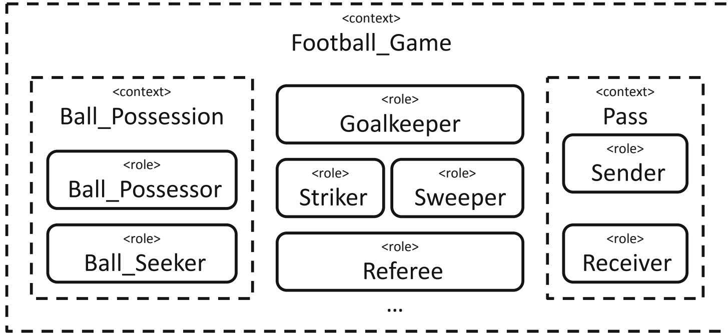


Fig. 2. Selected Concepts of Robot Soccer

In an inner context named `Pass` the special roles of players taking part in the pass collaboration (`Sender`, `Receiver`) are introduced.

Most of the logic behind soccer depends on the current game situation (i.e., on the current context of the collaborating robots). In a naive implementation, contexts and roles need to be represented using concepts of a conventional GPL. Here, we experience tangling code of different collaborations (i.e., behavior to be performed in different situations) and scattered code for a single collaboration across the code of the application by replication of if-statements.

Listing 1 shows a simplified code snippet, specifying the behavior for a robot. The behavior of the robot depends on whether it is in possession of the ball (`if (BALL_POSSESSION)`), whether it is in the role of a goalkeeper (`if (GOALKEEPER)`) and whether it is the sender or receiver of a pass (`if (SENDER)`, `if (RECEIVER)`). The checks for the ball possessor, sender and receiver roles need to be replicated in the else-branch of the check for the goalkeeper role.

```
1  if (GOALKEEPER) {
2    if (BALL_POSSESSION) {
3      if(SENDER) { throw_ball: nearest_free_player: this; }
4      else if(RECEIVER) {...}
5    }
6    else {...}
7  }
8  else {
9    if(BALL_POSSESSION) { /* replicated if structure */
10     if(SENDER) { shoot_ball: nearest_free_player: this; }
11     else if(RECEIVER) {...}
12   } else {...}
13   ...
14 }
```

Listing 1. Example Behavior Specification Including four Roles

The illustrated nesting is required, because the actual behavior in the branches could differ. As shown in the example, a goalkeeper will throw the ball to the nearest player relative to himself (cf. line 3), whereas a usual player has to shoot the ball (cf. line 10). Notably, each additional role and each additional context will further impair replication. Scattering leads to issues in code maintenance, as a change in the behavior of a goalkeeper requires adjustments of multiple—syntactically distributed and unrelated—code segments. Hence, suitable abstraction and modularization mechanisms are required to avoid code scattering and replication. Before we present NaoText in Sect. 4—which comprises such mechanisms—we will outline our architecture in the next section.

3 Applying SOA for Simple Robot Coordination

This section introduces our architecture for implementing the interpreter used to evaluate NaoText-based specifications of collaborative robot applications. Shifting the focus of robot programming from singular automation units to distributed, mobile teams of collaborating robots induces the need for a communication architecture that establishes communication channels among the involved robots [26]. We suggest a distributed architecture that uses Web services for communication. The numerous benefits of Web services and SOA introduced in [12] are beneficial for the following reasons:

Platform Independence. Web services provide a standardized communication protocol that is implemented in and can be accessed from various platforms. This is beneficial both for implementing services and the service interface for concrete robot platforms to remotely access and control these services.

Declarative Interface. Web services declare an interface abstracting the physical implementation of a given functionality. This means implementation can be exchanged, adopted or ported to other infrastructures. This is beneficial w.r.t. the heterogeneity found in today's robot platforms.

Location Transparency. Services are invoked through a communication network that routes service calls to the receivers independent of their location.

Various authors [14,16,18,24,25] suggest the implementation of SOAs for robot communication using the simple object access protocol (SOAP) or the common object request broker architecture (CORBA). Both approaches are often criticized for requiring a sophisticated messaging infrastructure. This is a problem, especially w.r.t. the restricted capabilities of software running on robots. We address this issue by implementing SOA using REST. As REST builds on HTTP and URIs for routing and calling Web services, it alleviates the need for custom communication middleware and can easily be deployed for robots connected through a wireless network. In the following, we describe the implementation of a REST-ful SOA for Nao robots.

Fig. 3 shows the application of SOA concepts to remotely control and coordinate a set of Nao humanoid robots. It consists of two stacks of layers.

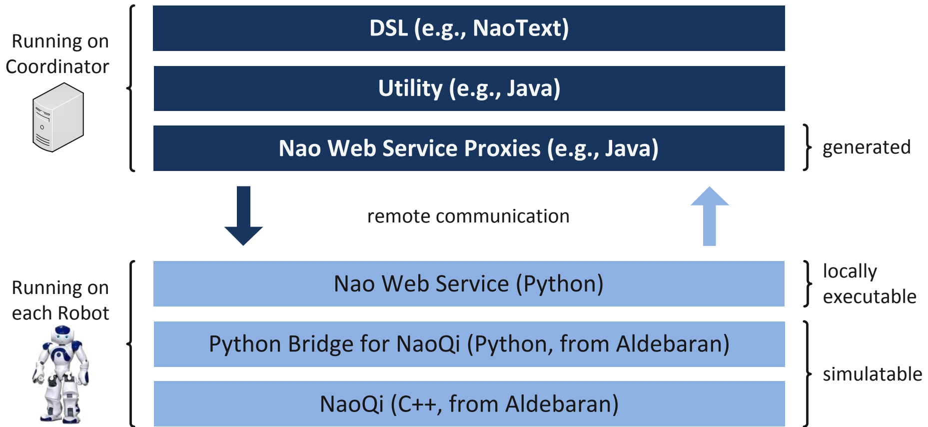


Fig. 3. Example for a layered architecture to control Nao robots

The lower stack contributes three layers implementing REST for Nao robots. The upper stack of layers runs on the *coordinator* that accesses this REST interface for controlling a collaborating team of robots (cf. Fig. 1).

Naos are controlled via NaoQi which represents the lowest layer of our architecture that is closest to the Naos' hardware (cf. Fig. 3). NaoQi provides an API for interacting with individual Nao sensors. Furthermore, it provides an API for basic actions such as controlling specific motors of a Nao or invoking the text-to-speech module. NaoQi is implemented in C++ and provided by Aldebaran. On top of NaoQi, Aldebaran provides a Python bridge that exposes the NaoQi API for the Python language. Using Python, we built a Web service layer that exposes the NaoQi services as REST-ful Web services. This Web service layer is automatically generated from the Python bridge using the introspection capabilities of Python. This approach strongly reduced our implementation effort and also eases the evolution of the Web service interface when NaoQi changes. Besides directly accessing the Web services, all provided services can be explored using a web browser. Each Web service can be called using HTML forms (generated by the Web service using templates). Our implementation is published[5] under GPL.

To access the Web service from the coordinator, we implemented a code generator that generates Java proxies. The proxies encapsulate the remote communication with the generated Web services. Above this layer, we implemented a number of utility interfaces that combine a number of low level Web services to more abstract behavior units. E.g., it provides a method `walkTo(int x, int y)` to let a Nao walk into a certain direction. Internally, the method computes the angle into which the robot has to rotate and how far it has to walk. Afterwards, a proxy object is created and the command is transmitted to the robot. Further above, in the top layer of our architecture, we find the interpreter for NaoText, which is described in more detail in Sect. 4. Briefly, the interpreter is executed by

[5] `http://code.google.com/p/naoservice/`

the coordinator and interprets NaoText programs. The interpreter also monitors the system status and manages the activation of contexts and bindings of roles to robots.

This simple SOA provides a communication infrastructure between our interpreter and the individual Nao robots. Similar solutions are easy to implement for other robot platforms. Further, the three lower layers can be simulated on desktop PCs to emulate the Nao robots. Thus, our architecture allows in-the-loop development and test of NaoText programs, too.

4 NaoText for Controlling Collaborating Robots

In this section, we introduce NaoText—a role-based DSL to specify the behavior of collaborating robots. It contributes an appropriate abstraction and ensures platform-independence for collaborative robot applications. We introduce the design of NaoText and illustrate its application on our motivational example.

4.1 Design of NaoText

In Fig. 4 we illustrate the design of NaoText by an excerpt of its metamodel. To manage the complexity of collaborative tasks we choose `Role` and `Context` as central abstractions in NaoText. The concept of roles, first applied over 30 years ago [2], embodies partial behavior of participants in a collaboration [19]. Thus, these concepts allow for concise specifications of collaborations. Roles are of founded, non-rigid types [15]. The property of being founded connotes that a type is dependent on another type. Consequently, `Role`s cannot exist on their own, but have to be contained in a `Context`. Non-rigidity connotes that instances do not cease to exist, when they stop having a non-rigid type. For example, a concrete robot being a striker does not cease to exist, when it stops being a striker. In consequence, each role needs to be bound to a player (e.g., the robot in the aforementioned example).

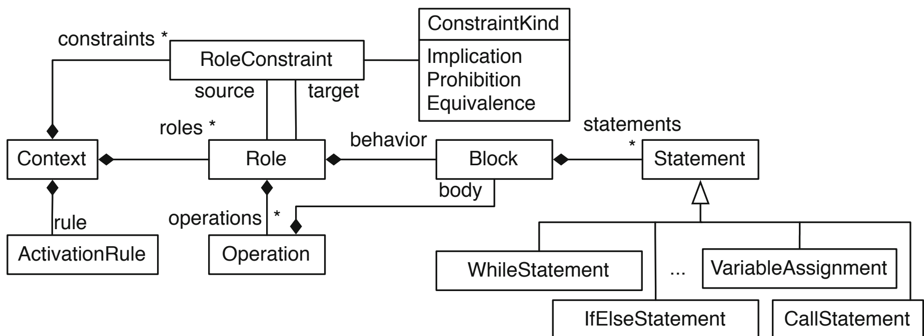


Fig. 4. Excerpt of Metamodel for the role-based NaoText DSL

A multitude of approaches to constrain this binding exist. Besides hard constraints, restricting the players of a role to a single type as in ObjectTeams [17], complex set constraints have been investigated in the context of first-class relationships [5]. NaoText uses a lean, but expressive way to constraint role bindings that was introduced in [21]. It allows the definition of three kinds of binary `RoleConstraint`s: `Implication`, `Equivalence`, and `Prohibition`. If a role A implies role B, every player of role A has to play role B in addition. The role-equivalence applies the implication in a bidirectional way. Finally, if a role A prohibits a role B, no player of A is allowed to play role B at the same time.

The binding of roles to players during runtime is controlled using `ActivationRules`. These rules describe patterns for potential role players and trigger the activation of a context if matching players are found. Such context activation imposes the binding or rebinding of the roles for the matched players and initiates the evaluation of their respective behavior.

To model the behavior of roles in NaoText, each `Role` contains a `behavior` `Block`. This `Block` contains a list of `Statement`s. The kinds of statements available in NaoText resemble those typically found in imperative, object-oriented languages (e.g., `WhileStatement`, `IfElseStatement`, `VariableAssignment`). Using `CallStatement`s, behavioral specifications can be modularized. `CallStatement`s can refer to `Operation`s or call Web services defined for Naos.

4.2 Application on Our Motivating Example

In the following, we will present the application of NaoText for our motivating example. First, we discuss how the declarative part of NaoText (roles and contexts) is used to structure robot collaborations. Then, we present the application of imperative language concepts.

Listing 2 shows the declaration of the `Pass` (lines 11–44) context as introduced in Fig. 2 in NaoText syntax. The context `Pass` introduces two roles: the `Sender` (lines 24–36) and the `Receiver` (lines 38–43). The context's activation is controlled by the activation rule defined in lines 14–22. The rule relates to roles defined in the surrounding context `Ball_Possession` (lines 1–45). Whenever a robot playing the role `BallPossessor` identifies another robot playing the role `BallSeeker` and is itself not able to do a shot on goal, the context `Pass` is activated. As a result of the activation, the `BallPossessor` becomes a `Sender` and the `BallSeeker` becomes a `Receiver`. Multiple such rules can be defined for each context. In addition, a role-prohibition constraint expresses the exclusiveness of the roles `Sender` and `Receiver` (line 12).

Besides the definition of contexts and roles, we use NaoText to describe the robots' behavior w.r.t. roles they are currently playing. Therefore, we use imperative language constructs to control services to be fulfilled by the robots. In Listing 2 both roles (`Sender` and `Receiver`) comprise a `behavior` block, which defines the roles' behavior (lines 26–34 and 39–42). The `Sender` role defines the behavior to be performed by a ball possessing robot, which intends to pass the ball. If the robot concludes that the ball is catchable by an opposing robot, it will feint a shoot and walk away. Else a pass is performed by passing the ball

```
1  context Ball_Possession {
2    role BallSeeker {
3      behavior {...}
4      void randomWalkWithBall {
5        walk_to: random_int(20), random_int(20)
6      }
7      //...
8    }
9    role BallPossessor {...}
10
11   context Pass {
12     Sender prohibits Receiver;
13
14     activate for {
15       BallPossessor p;
16       BallSeeker s;
17     when {
18       (p.robotInVision: s) and not (p as Striker).isGoalShotPossible;
19     } with bindings {
20       p->Sender;
21       s->Receiver;
22     }
23
24     role Sender {
25       attr passRatio: float;
26       behavior {
27         if(ballCatchableByOpponent) {
28           feintShoot;
29           randomWalkWithBall;
30         } else {
31           boolean hit = shootBall;
32           updatePassRatio: hit;
33         }
34       }
35       void updatePassRatio hit:boolean {...}
36     }
37
38     role Receiver {
39       behavior {
40         waitForBallInVision;
41         catchBall;
42       }
43     }
44   }
45 }
```

Listing 2. Example NaoText Code including the Pass Context

to another robot. Notably, the `Sender` role includes an attribute for the ratio of successful or failed passes, which can be used for a more sophisticated decision making procedure than in the example shown here. The `Receiver` role defines that the robot will wait for the ball to appear in its vision followed by catching the ball. After the ball has been catched by the `Receiver`, the context is finished and other context activation rules will trigger a reconfiguration of role bindings. At runtime, the NaoText interpreter evaluates the description of role behavior and invokes the `behavior` method of the respective roles. The individual instructions are evaluated and, finally, transmitted as calls to the robot playing the respective role using the communication infrastructure introduced in Sect. 3.

In this section, we applied NaoText to declare roles and contexts and to imperatively describe the roles' behavior for soccer playing. Of course, implementing a complete soccer scenario involves more contexts and roles as outlined above. Also the concrete behavior specification are likely to be more complex as well. However, we think that the example can give an impression of how NaoText can improve the development of collaborative robot applications. In the next section, we present an initial evaluation of NaoText by discussing its benefits and challenges w.r.t. alternative approaches to specify robot collaborations.

5 Discussion

Using a SOA comes together with both advantages and disadvantages in the context of collaborative robot applications. On the one hand, using a SOA instead of other mechanisms like remote method invocation (RMI), leads to communication overhead at runtime. Thus, we decided to use a REST-ful SOA to minimize this communication overhead. On the other hand, SOAs allow for location transparency and uniform addressability. Especially, our proxy generator for the Nao Web service provides easy connectivity for every GPL.

As our approach proposes the execution of NaoText programs on a central coordinator node, it can be considered as an approach that shifts the application logic from individual robots into the cloud. This leads to thin robots that propagate their state to the coordinator node and receive commands they have to execute to interact with their environment. This approach has advantages as well as disadvantages, too. Coordinating all participating robots from a central node leads to operability problems once the coordinator node fails. In this situation, the robots can neither communicate with each other nor collaborate. However—when operating—the central coordinator node maintains the global state of the application and thus, knows about the robots' locations and their further attributes (e.g., whether or not they are close to other robots etc.). Thus, the coordinator can easily grant and remove roles from individual robots to control the collaboration in an optimal way. Furthermore, the deployment of the application logic into the cloud avoids the computation of complex applications on the robots. Thus, they can save their often limited resources for basic functionality which leads to better robot operability (e.g., longer battery life time).

In addition of using a central coordinator node, we propose to use a role-based DSL which leads to the following advantages and disadvantages: First, of course,

the implementation of a new DSL imposes efforts such as the language's design and the implementation of a parser, an interpreter, and/or a compiler. However, afterwards, the DSL leads to several advantages in contrast to GPLs. A DSL allows for abstraction from general-purpose concepts and expressing the behavior in lesser instructions being more appropriate for the domain of the language (i.e., humanoid robots). For example, instead of writing a block of Java code comprising several statements to let a Nao walk to a certain position, a single statement is sufficient to express the same behavior within NaoText. Thus, DSLs lead to more intuitive behavior implementations that are easier to understand and maintain. Besides, by introducing imperative robot commands that abstract from interface invocations for robot-platform specific services, the DSL can be easily connected to other robots. Instead of writing a completely new NaoText program, we only have to create new service adapters for the newly introduced robot type (e.g., to integrate Mindstorm robots into our soccer scenario). Besides the advantages of DSLs in general, the role-based concepts integrated into NaoText allow for a declarative expression of robot collaborations. Code replication and sections of nested if-statements can be avoided. By this, the utilization of roles fosters comprehensibility and maintainability. A similar implementation in imperative source-code would lead to a robot's behavior scattered over many if-statements checking its current contexts and state.

Finally, the annotation and analysis of non-functional properties is vital for handling safety-, time-, or resource critical characteristics of robots interacting with the physical world. A DSL offers the right level of abstraction to easily implement static analysis and testing capabilities. E.g., role constraints provide an easy concept to prohibit behavior of robots that is expected to be not executed in parallel (e.g., a referee should not shoot a goal). Furthermore, the imperative statements of NaoText provide the right level of abstraction for the estimation of non-functional properties (e.g., execution time or energy consumption) of individual Naos, whereas by using a complete GPL for static analysis, large parts of the analysis would have to deal with several general-purpose concepts being less important for the context of collaborative robot applications.

6 Related Work

NaoText as presented in this paper contributes a communication architecture and a role-based DSL to specify the behavior of collaborative robot applications. In this section we discuss related work and differences to our approach.

Communication Aspects. The authors of [26] state that communication between robots is essential for team play and therefore discuss explicit communication as an approach for robot collaborations. Many works use SOA to address the problem of realizing the collaboration between robots. The works of [14,16,25] propose either to use CORBA or SOAP for the communication among robot teams. In contrast, we decided to use REST-ful services—like some platforms for networked mobile robotics do [6,11]—because REST does not need any additional

communication middleware as it has no additional transport layer. Furthermore, [24] proposes to use Web services to access and control robots, where a single machine orchestrates the synchronization between different tasks and robots. This infrastructure offers control over a group of heterogeneous robots through the Internet. Thus, they address the lack of standardized interfaces and communication protocols to interconnect heterogeneous robots over the Internet. In contrast to our approach, the Web service protocol was not implemented directly on the robots and no DSL to specify collaborations is presented. Chen and Bai describe another scenario with collaborative robots [8]. In their approach the robots are orchestrated by a remote collaboration center (RCC) and communicate via Web services. The *Robotics DeveloperStudio* from Microsoft is used to realize collaborations. To ease the communication between robots and to provide reusable software components, the *robot operating system (ROS)* framework has been developed [18]. It provides a communication layer for robot applications that can be used for both peer-to-peer communication and SOA-like communication between nodes in robot applications. Besides the communication layer, ROS provides a hardware abstraction layer and libraries for commonly used functionality. The framework is language independent in the way that it is available for multiple GPLs, like Phyton, C++ and Lisp. Experimental implementations for Java and Lua exist as well. The open-source framework *urbi*[6] can be used to control robots and their collaboration in general. Therefore it offers a low level C++ component library, which simplifies the process of writing programs. Multiple robot-specific implementations of urbi exists. As the resources for computing algorithms on robots themselves are limited, recent efforts have been spent to shift computationally intensive functions into the cloud. *DAvinCi* is such a cloud computing framework for collaborating service robots [1]. In DAvinci robotic algorithms are exposed as a service. The data is shared among the robot cluster cooperatively. The term *robot as a service (RaaS)* is created in [9]. The work shows, that a robot can be used in the cloud as a SOA-unit, providing and consuming services and acting as a service broker. Challenges for the development of distributed robotic services are summarized in [20].

Language Aspects. Using a DSL to describe robot behavior and collaboration is promising, since it helps to abstract from hardware and low level implementation details. *Spica* is a model-driven software development environment for creating multi-robot communication infrastructures [3,4]. The development environment consists of several DSLs to specify the different aspects of robot communication. In contrast to our approach, the DSL used to describe robotics' behavior is not capable of roles. Naos are delivered together with the development environment *Choregraphe*. It allows developing a Nao's application in a graphical and component-based manner. Complex applications can be plugged together using basic building blocks like *stand up*, *sit down*, or *text-to-speech*. Although Choregraphe allows easy development of applications for single Nao robots, it does not support Nao interaction nor interaction between Naos and other robots

[6] http://www.urbiforge.org/

or players in its environment, neither in an imperative nor in a declarative way. The *Robotics DeveloperStudio* from Microsoft provides a runtime environment to create, host, manage, and connect distributed services. The language used in the Robotics DeveloperStudio is called *VPL* . It builds on the .NET framework and is comparable to the language used in Choregraphe.

Roles for Collaborative Robot Applications. In our approach, we propose to use roles to describe the collaboration between robots. The applicability of roles in controlling cooperative robots is examined in [7]. The robots act independently, but can react to messages they send to each other. The concept of roles as used in [7] is comparable to a state the robot is in at a time. Therefore, roles and role changes are expressed as a finite automaton for each robot. Our example introduced in Sect. 2 shows the ability of a robot to play multiple roles at once. In consequence, modeling role changes as transitions in an automaton leads to an exploding number of states. This is because each possible combination of roles has to be represented by a separate state of the automaton. In contrast, we declare role and context changes by means of activation rules as shown in Listing 2. An approach using roles for the collaboration of modules a robot consists of is presented in [22]. In this approach, roles are used to express the (active and reactive) behavior as well as the structure of the modules of a robot. The authors provide a role-based DSL called *Role-based Distribution Control Diffusion (RDCD)* to implement their approach for the ATRON self-reconfigurable robot. The language is an extension to *RAPL (Role-based ATRON Programming Language)*, that is presented in [10]. Role changes can occur as a reaction of a message from one of the robots' modules or internal events send by sensors of a robot. The DSL provides primitives for making simple decisions, whereas complex computations are described externally. They focus on a single robot and the collaboration between its modules, where only neighbor modules are able to collaborate. In contrast, our approach focuses on the collaboration of multiple independent robots.

7 Conclusion

In this paper we discussed the fundamental concepts of NaoText, a DSL supporting role concepts for the implementation of collaborative robot applications. We presented an architecture to evaluate NaoText programs based on a central coordinator that monitors the system status and controls collaborative robot teams. For communication between the coordinator and the individual robots this architecture uses REST-ful Web services. We demonstrated the application of NaoText using an exemplary football application. Our approach leverages abstraction and comprehensibility in the implementation of collaborative robot behavior and contributes a lightweight architecture for collaborations among robots that is easily extensible to support further robot platforms besides Naos. We discussed differences and commonalities of our approach with related work.

For future work, we plan to implement an interpreter for NaoText and to complement the NaoText tooling with static analyzers, to ensure functional and

non-functional properties (NFPs) (e.g., performance and energy consumption) and to develop testing capabilities for NaoText applications. We also plan to improve the expressiveness of NaoText by integration of predicate dispatch [13] (i.e., the dynamic selection of appropriate method implementations w.r.t. predicates over the state, structure and NFPs of the system). Notably, the currently used role dispatch is a special case of predicate dispatch. Finally, we will extend our interpreter architecture to support further robot platforms and other physical devices, to enable the specification of sophisticated cyber-physical systems.

Acknowledgement. This research is funded by the DFG within CRC 912 (HAEC), the European Social Fund and Federal State of Saxony within the project ZESSY #080951806 and by the European Social Fund, Federal State of Saxony and SAP AG within project #080949335.

References

1. Arumugam, R., Enti, V., Bingbing, L., Xiaojun, W., Baskaran, K., Kong, F.F., Kumar, A., Meng, K.D., Kit, G.W.: DAvinCi: A cloud computing framework for service robots. In: IEEE International Conference on Robotics and Automation (ICRA 2010), pp. 3084–3089 (2010)
2. Bachman, C., Daya, M.: The role concept in data models. In: Proceedings of the 3rd Conference on Very Large Data Bases (VLDB), pp. 464–476 (1977)
3. Baer, P.A., Reichle, R.: Communication and Collaboration in Heterogeneous Teams of Soccer Robots, ch.1, pp. 1–28. Tech Education and Publishing, Vienna (2007)
4. Baer, P.A., Reichle, R., Geihs, K.: The Spica Development Framework - Model-Driven Software Development for Autonomous Mobile Robots. In: Proceedings of International Conference on Intelligent Autonomous Systems (IAS 2010), pp. 211–220. IAS Society (2008)
5. Balzer, S., Gross, T.R., Eugster, P.T.: A Relational Model of Object Collaborations and Its Use in Reasoning About Relationships. In: Bateni, M. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 323–346. Springer, Heidelberg (2007)
6. Cardozo, E., Guimaraes, E.G., Rocha, L.A., Souza, R.S., Paolieri Neto, F., Pinho, F.: A platform for networked robotics. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), Taiwan, pp. 1000–1005 (2010)
7. Chaimowicz, L., Campos, M., Kumar, V.: Dynamic role assignment for cooperative robots. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 293–298. IEEE (2002)
8. Chen, Y., Bai, X.: On Robotics Applications in Service-Oriented Architecture. In: 28th International Conference on Distributed Computing Systems Workshops (ICDCS 2008), pp. 551–556 (2008)
9. Chen, Y., Du, Z., Garcia-Acosta, M.: Robot as a Service in Cloud Computing. In: IEEE International Workshop on Service-Oriented System Engineering, pp. 151–158 (2010)
10. Dvinge, N., Schultz, U., Christensen, D.: Roles and Self-Reconfigurable Robots. In: Roles 2007, pp. 17–26 (2007)

11. Edwards, R., Parker, L.E., Resseguie, D.R.: Robopedia: Leveraging Sensorpedia for Web-Enabled Robot Control. In: Proceedings of 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM 2010), pp. 183–188 (2010)
12. Erl, T.: Service-oriented architecture: Concepts, Technology, and Design. Prentice Hall PTR (2005)
13. Ernst, M., Kaplan, C., Chambers, C.: Predicate Dispatching: A Unified Theory of Dispatch. In: Jul, E. (ed.) ECOOP 1998. LNCS, vol. 1445, pp. 186–211. Springer, Heidelberg (1998)
14. Flückiger, L., To, V., Utz, H.: Service-Oriented Robotic Architecture Supporting a Lunar Analog Test. In: International Symposium on Artificial Intelligence, Robotics, and Automation in Space, iSAIRAS (2008)
15. Guarino, N., Welty, C.A.: A Formal Ontology of Properties. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS (LNAI), vol. 1937, pp. 97–112. Springer, Heidelberg (2000)
16. Ha, Y.-G., Sohn, J.-C., Cho, Y.-J.: Service-Oriented Integration of Networked Robots with Ubiquitous Sensors and Devices using the Semantic Web Services Technology. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 3947–3952 (2005)
17. Herrmann, S.: A Precise Model for Contextual Roles: The Programming Language ObjectTeams/Java. Applied Ontology 2(2), 181–207 (2007)
18. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
19. Reenskaug, T., Wold, P., Lehne, O.: Working with objects - The OOram Software Engineering Method. In: TASKON (1995)
20. Remy, S.L., Blake, M.B.: Distributed Service-Oriented Robotics. IEEE Internet Computing 15, 70–74 (2011)
21. Riehle, D., Gross, T.: Role model based framework design and integration. In: Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 1998), pp. 117–133. ACM, New York (1998)
22. Schultz, U.P., Christensen, D.J., Stoy, K.: A domain-specific language for programming self-reconfigurable robots. In: Workshop on Automatic Program Generation for Embedded Systems (APGES), pp. 28–36 (October 2007)
23. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. IEEE Transactions on Data and Knowledge Engineering 35(1), 83–106 (2000)
24. Trifa, V.M., Cianci, C.M., Guinard, D.: Dynamic Control of a Robotic Swarm using a Service-Oriented Architecture. In: 13th International Symposium on Artificial Life and Robotics, AROB 2008 (2008)
25. Wu, B., Zhou, B.-H., Xi, L.-F.: Remote multi-robot monitoring and control system based on MMS and web services. Industrial Robot: An International Journal 34, 225–239 (2007)
26. Yokota, K., Ozaki, K., Watanabe, N., Matsumoto, A., Koyama, D., Ishikawa, T., Kawabata, K., Kaetsu, H., Asama, H.: UTTORI United: Cooperative Team Play Based on Communication. In: Asada, M., Kitano, H. (eds.) RoboCup 1998. LNCS (LNAI), vol. 1604, pp. 479–484. Springer, Heidelberg (1999)

Efficient Localization for Robot Soccer Using Pattern Matching

Thomas Whelan, Sonja Stüdli, John McDonald, and Richard H. Middleton*

Department of Computer Science, NUI Maynooth, Maynooth, Co. Kildare, Ireland
Hamilton Institute, NUI Maynooth, Maynooth, Co. Kildare, Ireland
{thomas.j.whelan,sonja.stuedli,richard.middleton}@nuim.ie,
johnmcd@cs.nuim.ie

Abstract. One of the biggest challenges in the RoboCup Soccer Standard Platform League (SPL) is autonomously achieving and maintaining an accurate estimate of a robot's position and orientation on the field. In other robotics applications many robust systems already exist for localization such as visual simultaneous localization and mapping (SLAM) and LIDAR based SLAM. These approaches either require special hardware or are very computationally expensive and are not suitable for the Nao robot, the current robot of choice for the SPL. Therefore novel approaches to localization in the RoboCup SPL environment are required. In this paper we present a new approach to localization in the SPL which relies primarily on the information contained within white field markings while being efficient enough to run in real time on board a Nao robot.

1 Introduction

In our earlier work [12] we gave some initial thoughts and results for an algorithm based on Cox's algorithm (a form of least squares error pattern matching) for effective robot self localization based on field markings. As we continued to use this algorithm, we became aware of a number of significant shortcomings in the algorithm in the form proposed in [12] and in addition, were able to introduce a number of additional features suggested in this previous work. The main aim of this paper is to address some of the issues highlighted in our previous work and describe our progress with some of the future work mentioned therein [12]. This paper is also intended to function as a stand alone reference document for those who wish to implement this approach to localization themselves.

Previous to any RoboCup competition a complete description of the RoboCup SPL field is provided for competing teams [8]. As can be seen in Figure 1, this is a wholely static environment with a large amount of concise visual information including field lines, goal posts, penalty spots and the centre circle. During soccer matches some dynamic elements do present themselves such other robots,

* This work was supported by Science Foundation Ireland, PI Grant no. 07/IN.1/I1838 and UREKA Site Grant no. 09/UR/I1524.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 16–30, 2012.
© Springer-Verlag Berlin Heidelberg 2012

referees and audience members beyond the pitch boundaries. A typical localization system which may used in a RoboCup environment is described by Röfer et al. in [9].

Some of the challenges associated with using white field markings in localization include the development of robust vision algorithms for line detection and line fitting, as well as techniques for identifying circular field markings and penalty spot type features. Given the small field of view provided by the camera on board the Nao robot, oftentimes only small segments of white field markings are visible in any given image. This ultimately leads to the difficult problem of uniquely identifying a certain field marking from a partial view. An immediate issue with common solutions to this problem is the resultant spaghetti code - large blocks of nested conditional statements tailored specifically to the RoboCup pitch layout are required. The solution we present removes the need to design ad-hoc line detection and identification systems by functioning generically on any combination of simple geometric features.

Fig. 1. Standard Platform League Pitch Description [8]

2 Background

Our method for utilising field markings is a specialised extension of matching detected 'line points' to a predefined map of line segments. This algorithm, in the context of laser scanner based analysis, was first used by Cox in 1991 [1]. Cox treated the problem as least-squares linear regression with an analytical solution, successfully demonstrating that this technique was both accurate and practical. Lauer et al. described a similar algorithm in 2006 for robots in the RoboCup Midsize League using points detected on white field markings instead of laser points [5]. They also introduced a new error function with a numerical gradient descent based solution. In 2010, Rath implemented an adaptation of Lauer et al's algorithm for use on the Nao robot in the SPL [7]. This adaptation emulated most of the methods used by Lauer et al. but due to the hardware constraints of the Nao some small changes were introduced.

2.1 Previous Work

In our previous paper we described a modified version of Cox's original algorithm dubbed the Modified Cox Algorithm (MCA) [12]. We presented a number of modifications to the original algorithm including (i) the use of a Voronoi diagram to reduce computational load in determining the nearest field marking to a given point; (ii) an extension of the basic algorithm to include all types of field markings (line segments, circles and single points); (iii) distance based outlier detection; and, (iv) weighted least-squares cost minimization. We also detailed the integration of the MCA with an Unscented Kalman Filter. For the sake of completeness the MCA which we described previously is listed in the following section in its entirety, excluding the Kalman Filter integration. Our new improved Kalman Filter integration is described in detail in Section 3.1.

2.2 Modified Cox Algorithm

This description is a combination of the original method described by both Cox and Rath and the modifications we presented in our previous paper [1,7,12]. Given a set of detected points on white field markings in an image, the basic process of the Modified Cox Algorithm involves 3 main steps;

2.2.1 Point Transformation from Image to World Coordinates

Transformation from image coordinates to world coordinates is achieved using typical back projection associated with the extrinsic and intrinsic camera parameters (see for example Figure 2a). In this regard the camera location is based on the geometry of the robot and the current joint sensor readings.

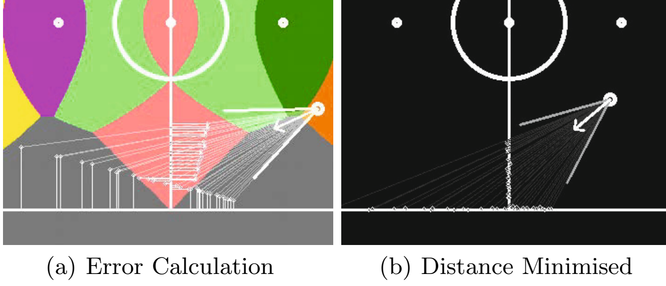
2.2.2 Selecting the Closest Field Marking to Each Point

Before this step is carried out a Voronoi diagram for all white field markings must be pre-calculated, see Figure 2 (a). This can be done once off during the start up of the robot. Then, the closest white field marking to any point projected into world coordinate space can be determined in $O(1)$ time.

2.2.3 Finding a Correction for the Current Pose

In this final step, a correction to the current robot pose, described by $l_t = (x, y, \theta)^\top$ where x and y describe the estimate of the robot's global position and θ describes the estimated orientation of the robot, is calculated. We wish to calculate $b = (\Delta x, \Delta y, \Delta\theta)^\top$ such that a new estimate, $l'_t = l_t + b$, gives a pose which better matches observed points to white field markings.

The aim of Cox's original algorithm is to minimise the squared distances associated with points on line segments (line points) to their nearest line segment. To achieve this, the problem is linearised into a least-squares linear regression problem and each line segment is treated as an infinite line with orthogonal unit vector $u_i = (u_{ix}, u_{iy})^\top$ and offset r_i such that $u_i \cdot z_i = r_i$ holds for all arbitrary line points z_i on the line.



(a) Error Calculation (b) Distance Minimised

Fig. 2. Example of the Modified Cox Algorithm

Let the ith transformed line point be $z_i = (z_{ix}, z_{iy})^\top$ and the current position of the robot be $c = (l_{tx}, l_{ty})^\top$. The transformation of each line point z_i can be described as:

$$t(b)(z_i) = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix} (z_i - c) + c + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{1}$$

Cox suggests that the correction angle $\Delta\theta$ should be sufficiently small such that we can approximate the transformation to:

$$t(b)(z_i) \approx \begin{pmatrix} 1 & -\Delta\theta \\ \Delta\theta & 1 \end{pmatrix} (z_i - c) + c + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{2}$$

Next, the squared distance of each line point z_i can be found as:

$$d_i^2 = (t(b)(z_i)^\top u_i - r_i)^2 \approx ((x_{i1}, x_{i2}, x_{i3})b - y_i)^2 \tag{3}$$

Where:

$$\begin{pmatrix} x_{i,1} & x_{i,2} & x_{i,3} \end{pmatrix} = \left(u_{ix} \;\; u_{iy} \,\middle|\, u_i^\top \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (z_i - c) \right) \tag{4}$$

$$y_i = r_i - z_{ix}u_{ix} - z_{iy}u_{iy} \tag{5}$$

Defining the absolute fixed world position of a penalty spot as $s_i = (s_{ix}, s_{iy})^\top$ and $q_i = (q_{ix}, q_{iy})^\top$ as a transformed penalty spot point we have:

$$\begin{pmatrix} x_{i,1} & x_{i,2} & x_{i,3} \\ x_{i+1,1} & x_{i+1,2} & x_{i+1,3} \end{pmatrix} = \left(\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \,\middle|\, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (q_i - c) \right) \tag{6}$$

$$\begin{pmatrix} y_{i1} \\ y_{i2} \end{pmatrix} = \begin{pmatrix} s_{ix} - q_{ix} \\ s_{iy} - q_{iy} \end{pmatrix} \tag{7}$$

Assuming a centre of $(0, 0)$ for the centre circle feature, given a radius of h and a transformed centre circle point $v_i = (v_{ix}, v_{iy})^\top$ we have:

$$\begin{pmatrix} x_{i,1} & x_{i,2} & x_{i,3} \end{pmatrix} = \left(\frac{v_{ix}}{||v_i||_2} \;\; \frac{v_{iy}}{||v_i||_2} \,\middle|\, \frac{v_i^\top}{||v_i||_2} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (v_i - c) \right) \tag{8}$$

$$y_i = h_i - v_{ix}u_{ix} - v_{iy}u_{iy} \tag{9}$$

Since we expect smaller errors in the location of points close to the robot, we define a diagonal weighting matrix, W, with diagonal elements given as:

$$W_{ii} = \frac{1}{\alpha_i^2 + \eta}, \tag{10}$$

and where α is the relative distance to the point from the robot and η is some small offset value.

Now we can calculate the weighted sum of squared distances for all points z_i, q_i and v_i:

$$E_W(b) = \sum_{i=1}^{n} W_{ii}((x_{i1}, x_{i2}, x_{i3})b - y_i)^2 = (Xb - Y)^\top W(Xb - Y) \tag{11}$$

Where:

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & x_{n3} \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{12}$$

The correction $\hat{b}$ that minimises $E_W(b)$ can then be (approximately) solved by:

$$\hat{b} = (X^\top WX + \zeta I)^{-1} X^\top WY \tag{13}$$

where ζ is a small positive constant to avoid singularity occuring in (13). Finally a new pose is given by:

$$l'_t = l_t + \hat{b} \tag{14}$$

The results of this process can be seen in Figure 2b.

3 Extensions and Modifications

The main short comings of our previous implementation were localization performance and computational performance. We showed that the system was at most on par with three other localization systems evaluated and not significantly better by any measure [12]. Computational performance was also undesirably low for use on the Nao system, considering that localization is just one of many components that ideally must run at 30Hz during a soccer game. In this section we describe the extensions and modifications we made to the original system that aided in overcoming these issues.

3.1 Unscented Kalman Filter Integration

Initial tests with the algorithm described in [12] did not show good performance, and the CPU requirements were excessive. For these reasons, the algorithm we used in the RoboCup 2011 competition (including the technical challenge) underwent a major redesign. This redesign reduced the number of sigma points used and the number of iterations in the local search, and also added some extra computations to give a better estimate of the errors in the predictions from Cox's algorithm. The algorithm used is described below.

1. **Generation of a Set of Sigma Points**
The sigma points were calculated in a similar way to those of [3,11]. We have $n_x = 3$ state variables in the filter. In our case we selected the primary weight as $w_0 = \frac{1}{\sqrt{2}}$, and the remaining weights are selected as:
$$w_\ell = \sqrt{\frac{1 - w_0^2}{2n_x}}; \quad \ell = 1, ...2n_x. \tag{15}$$
We denote the current state estimate by $x_{k|k-1}$ and the ℓth column of the square root of the covariance matrix as $(p_{k|k-1})_{(\ell)}$. The sigma points, $\mathcal{X}_\ell$, are then calculated as:
$$\mathcal{X}_\ell = \begin{cases} x_{k|k-1} : \ell = 0 \\ x_{k|k-1} + s_\sigma (p_{k|k-1})_{(\ell)} : \ell = 1...n_x \\ x_{k|k-1} - s_\sigma (p_{k|k-1})_{(\ell)} : \ell = n_x + 1...n_x \end{cases} \tag{16}$$
where the scale factor for the sigma points is given by $s_\sigma = \sqrt{\frac{n_x}{1-w_0^2}}$.
2. **Use Each Sigma Point as an Initial Value for the Modified Cox Algorithm**
The Modified Cox Algorithm (MCA) described in Section 2.2 can be thought of as a local search for a good fit between the observed points and the known field markings. We use the sigma points in (16) as initial values for the MCA and perform a single step correction for each. The result of this update is a new estimate, $\hat{\mathcal{X}}_\ell$, of a possible robot location, together with the weighted sum of residual errors, $J_\ell = \sum_i W_{ii} e_i^2(\hat{\mathcal{X}}_\ell)$, and a covariance of the estimate, $var\hat{\mathcal{X}}_\ell$, computed as described below in (21).
3. **Discard Points with Excessive Error, or Other Problems**
The result of a single correction to one of the sigma points may not give a good fit to the data. This may be because a single iteration is insufficient to be close to convergence. It may also occur due to being close to a local minimum. We therefore select a threshold, $\bar{J}$, and ignore any results where the residual errors are too large, namely, $J_\ell > \bar{J}$. Other checks used are to test if the MCA has enough valid points to process and that the resultant estimated robot position is not too far off the pitch. Note that in the following equations, discarding a point is equivalent to setting the corresponding weight to zero.
4. **Adjust Sigma Point Weights According to the Residual Errors and Renormalize**
Given the initial set of weights, w_ℓ, we first adjust these according to the residual errors:
$$\tilde{w}_\ell := w_\ell / J_\ell. \tag{17}$$
We then renormalize the adjusted weights as follows:
$$\hat{w}_\ell := \tilde{w}_\ell / \sum_k \tilde{w}_k \tag{18}$$

5. **Check for Sufficient Valid Estimates**
For the algorithm to generate a valid measurement, we require that a sufficient number of sigma points had valid MCA results (in our case, this was set to 3), otherwise the entire set is ignored.
6. **Recombine Sigma Points into a Single Combined Estimate**

$$X = \sum_{\ell} \hat{w}_{\ell} \hat{\mathcal{X}}_{\ell} \tag{19}$$

$$var(X) = \sum_{\ell} \hat{w}_{\ell} \left[(\hat{\mathcal{X}}_{\ell} - X)(\hat{\mathcal{X}}_{\ell} - X)^T + var\hat{\mathcal{X}}_{\ell} \right] \tag{20}$$

7. **Use the Combined Estimate as a Linear Covariance Intersection KF Measurement Update**
The combined estimate, (19), is a linear function (in fact the identity) times the localization state variables. It is therefore straightforward to use linear Kalman Filter covariance intersection updates (e.g. [10,4]) to perform a measurement update. This update includes standard features such as outlier detection and kidnapped robot detection.

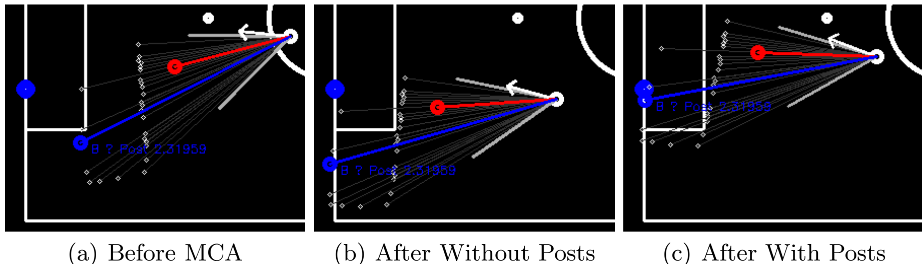
3.2 Modified Cox Algorithm

A number of additions and alterations have been made to the MCA. These have been divided into two categories: (i) Algorithmic Enhancements, concerned with the information used and produced by the algorithm; and, (ii) Computational Optimisations, concerned with the way in which the algorithm processes input and produces an output.

3.2.1 Algorithmic Enhancements

Super-Weighted Posts - The goal posts are one of the key sets of landmarks on the SPL pitch. They are generally quite easy to identify and are the most significant cue for localization. Having previously ignored any goal post information in the MCA optimisation, we now include goal posts in the form of super-weighted single points, akin to penalty spot type features.

Being color coded, the posts on the SPL field are inherently less ambiguous than points on white field markings. In line with this, post points are weighted 30 times higher than white field marking points in the optimisation but not in the error calculation in order to avoid inconsistencies in error values between frames including posts and those not. When both posts of a single color goal are visible a robot's position and orientation can be accurately determined using simple triangulation. In this scenario, both post points are matched to the known fixed positions of the posts in the world model for the MCA algorithm. More often than not however, only one post is visible due to optical occlusion by other robots. When this occurs the single visible post is ambiguous. Given that the MCA is a local search to begin with, an ambiguous post is matched to the fixed



(a) Before MCA (b) After Without Posts (c) After With Posts

Fig. 3. Example of Post Point Inclusion in MCA

position of the nearest post in the world model when the perceived post point is transformed into world coordinates.

An example is shown in Figure 3 of the kind of effect this feature has on the algorithm. In situations where the robot is mislocalized badly or a poor measurement to a post is perceived, there is concern for ambiguous posts being matched incorrectly and furthering corruption of the localization estimate. Typically in this scenario the system is more reliant on the Kalman Filter discussed in Section 3.1, which would normally have a high uncertainty when mislocalized and as such will have a large spread of sigma points and be more likely to throw out badly matched MCA updates.

Calculation of MCA Variance - In order to improve the integration of MCA updates in the Kalman Filter the variance of the calculated correction is recorded, for each sigma point. This is especially useful in scenarios where only points on co-linear field markings are detected. The variance of the translation and rotation is derived from the diagonal from the inverted component P of the final correction calculation listed as (13) in Section 2.2.3:

$$P = (X^\top W X + \zeta I)^{-1} \sigma_n^2 \tag{21}$$

The noise variance measurement is taken as $\sigma_n^2 = 0.01$.

3.2.2 Computational Optimisations

Point Sampling - Given that the MCA is highly dependent on matrix methods, a large amount of attention was given to these methods when optimising the technique for computational performance. Notably, for two matrices of dimensions $m \times p$ and $p \times n$ the run time complexity of standard matrix multiplication is $\mathrm{O}(mnp)$. The only input of varying size to the MCA is the number of points on white field markings. Owing to the complexity of the multiplication method it was observed that the execution time scaled badly. As a result, the number of white field marking points used in the MCA is capped at 30. If there are more points than this detected in an image 30 points are selected at random for use in the optimisation. The same 30 points are used for all 7 sigma points.

Matrix Operation Optimisation - As mentioned in Section 2.2.3, a diagonal matrix W is used to weight the optimisation. Rather than using naïve standard matrix multiplication in the evaluation of equations which involve the diagonal matrix W a more efficient multiplication method is used. This alternative method skips the summation step of standard matrix multiplication, which when used on a diagonal matrix would result in many redundant sums of zero. For two matrices (one diagonal) of dimensions $m \times p$ and $p \times n$ this alternative method runs in $\mathrm{O}(mn)$ time as opposed to $\mathrm{O}(mnp)$.

Another concern when optimising matrix operations is matrix chain multiplication. When presented with a sequence of matrices we wish to determine the most efficient way to multiply these matrices together, given that the parenthesization of matrix multiplication changes only the number of operations and not the result. Evaluating (13) in Section 2.2.3 in standard left to right order involves roughly 6000 operations when 30 points are used. Solving the matrix chain multiplication problem on this equation results in a decrease of the number of operations to approximately 3900. This represents a performance increase of 35%. The order of multiplication was determined using a dynamic programming approach [2]. The resulting parenthesization of (13) is:

$$\hat{b} = ((X^\top(WX)) + \zeta I)^{-1}(X^\top(WY)) \tag{22}$$

One final optimisation involves the precalculation of point weights. Rather than recalculating the weights for each sigma point, the weights of all points are calculated before hand as they do not change between sigma points. Overall a performance increase of 60% was recorded per sigma point with the MCA, derived from an execution time reduction of 1ms to 0.4ms per point.

In the table below execution time differences are shown for each of the described computational optimisations. The time given represents the increase in execution time of the entire MCA if that feature isn't implemented in the computation.

Optimisation	**Time Saved (ms)**
Reduction from 13 to 7 sigma points	6.1
Maximum of 30 line points	3.1
Diagonal Multiplication	1.2
Chain Multiplication	1.0
Weight Precalculation	0.1

3.3 White Field Marking Point Filtering

One concern highlighted by Rath was the issue of false positive points detected on white features on the RoboCup pitch [7]. This was less of a concern for Lauer et al. due to the fact that the robots used in the Midsize League do not contain white features. The Nao robot, used in the SPL, is almost entirely white and as a result false positive white field marking points are prevalent when color reliant detection algorithms are used.

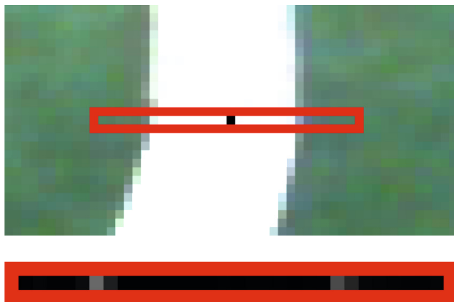


Fig. 4. Result of a Horizontal First Difference Operation Around a Point

In order to combat this we have implemented a filter based on edge detection principles that reliably removes false positives. When a set of points are detected in an image a local edge check is performed either horizontally or vertically depending on which scan orientation a point was detected with. As can be seen in Figure 4 there should exist a very obvious single maximum in a simple gradient estimate on both sides of a point. Inspecting both sides of a point for this maximum is a trivial process. A threshold can then also be applied to accept points with a large gradient.

Calculation of the gradient estimate is carried out using Intel MMX SIMD instructions in order to maintain good computational performance. The estimated gradient value for a given pixel (i, j) in the raw YUV image F is calculated using only the Y component as $|F_y(i, j) - F_y(i+1, j)|$ for horizontally detected points and similarly as $|F_y(i, j) - F_y(i, j+1)|$ for vertically detected points. These values can be calculated for 7 pixels in one go by using the technique described below.

3.3.1 MMX Gradient Estimation

A two pass technique is required to calculate the full first difference for a set of pixels using this method. The 64-bit MMX register is divided up into 8 unsigned integers each with a range of [0 - 255]. Given that the Y component of each pixel also has a range of [0 - 255] we are unable to use the MMX registers in a signed format. Thus, using the horizontal direction as an example, we calculate $F_y(i, j) - F_y(i+1, j)$ and $F_y(i+1, j) - F_y(i, j)$ separately using saturated arithmetic (clipping at 0 and 255) and sum the result together.

When processing points on white field markings however, the expected gradient direction is known and as a result we can skip first difference calculation in one direction.

Fig. 5. Sample Y Component Values at Point Surrounding

1. Populate an MMX register with the Y component values of the pixels around the edge of the detected point (at most 8). The pixels shown in Figure 5 would translate to an MMX register as:

140	142	144	160	235	240	240	240

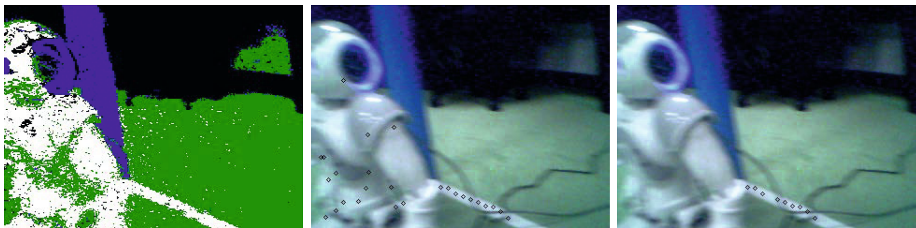
2. Depending on the expected gradient direction, based either on horizontal or vertical orientation and whether inspecting the edge going from green to white or vice-versa, shift a copy of the register left or right by 8 bits. In the example case, we shift right:

0	140	142	144	160	235	240	240

3. Using saturated MMX arithmetic, subtract the shifted register from the original, yielding the first difference for 7 pixels (we ignore the end value):

	140	142	144	160	235	240	240	240
-	0	140	142	144	160	235	240	240
=	-	2	2	16	75	5	0	0

The result of this calculation can then be inspected for a single maximum (with leniency) above a preset threshold. The MMX implementation described above was measured to be 2.6 times faster than a standard C++ implementation of the same process. Figure 6 shows an example of the results of this process. Occasionally good points are filtered out, particularly those that appear on features which are small in the image. Loss of such distant points is only a minor inconvenience due to the fact that they are weighted quite lightly in the MCA optimisation.

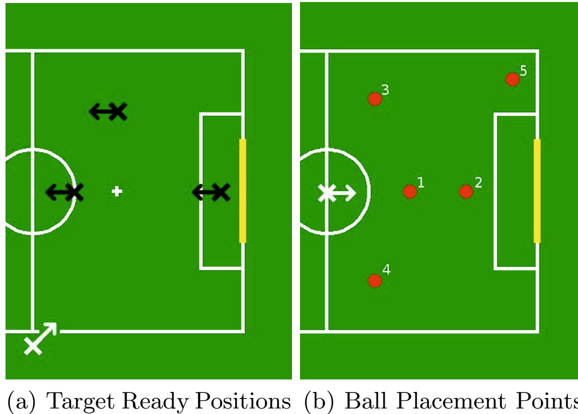


(a) Color Classified Image (b) Points Before Filtering (c) Points After Filtering

Fig. 6. Example of Detected Points With / Without Edge Filtering

4 Tests and Results

The performance of the newly modified MCA based localization system was evaluated in 3 different tests along with 3 other localization systems. The three tests were:



(a) Target Ready Positions (b) Ball Placement Points

Fig. 7. Test Scenario Set-Ups. Initial Position and Orientation Drawn in White.

1. **Ready Positions.** As shown in Figure 7 (a). For this test, the robot is placed in the initial position and commanded to walk to each of the positions marked in black as soon as a correct estimate of the initial position is reported by the localization system. This test was repeated 3 times for each position and once on each side of the field, bringing the total number of runs to 18.
2. **Open Goal Shots.** As shown in Figure 7 (b). In this test the robot is placed in the initial position and commanded to take shots on an open goal, again only when an accurate estimate of the initial position is reported. The ball is first positioned at the location labeled '1'. After each attempted shot on the goal the ball is placed at the next position in the labeled sequence. This test was carried out once on each side of the field giving a total of 10 attempted shots.
3. **Goalless Open Goal Shots.** This test is identical to the previous test except both color goals are removed from the field before the robot begins. Initially all goal posts are on the field to allow the robot acquire an estimate of its initial position. Once the robot acquires the correct initial position all goal posts are removed. As before this test was carried out once on each side of the field giving a total of 10 attempted shots.

The localization systems compared are as follows:

– **Kalman Filter + MCA:** The new localization system presented in this paper including MCA updates, post updates, line and corner updates.
– **Particle Filter + White Field Marking Points:** An experimental implementation developed for testing purposes. A basic Particle Filter using 100 particles using post updates, line updates, corner updates and a new experimental update that uses certain functions of the MCA. This update involves steps 1 and 2 in Section 2.2 followed by evaluation of the error magnitude of the projected points. The optimisation in step 3 is not carried out. Instead, the error before the optimisation is used to weight particles.

- **Particle Filter + Lines:** A basic Particle Filter using 100 particles, post updates, line updates and corner updates.
- **Kalman Filter + Lines:** The same Unscented Kalman Filter used with the MCA in this paper using only post updates, line updates and corner updates.

4.1 Localization Performance

Figure 8 shows the results of all 3 tests. For the Ready Position test, the percentage of times the robot successfully reached the target position is given. For each Ready Position test, the robot's final resting position and orientation was manually recorded. The robot was deemed unsuccessful if it either left the field or halted with a position which was greater than 30cm from the target location or an orientation which was greater than 15 degrees from the target orientation. In some cases the robot would never halt completely and oscillated around some final location, however there was no penalty for this behaviour if the position was correct.

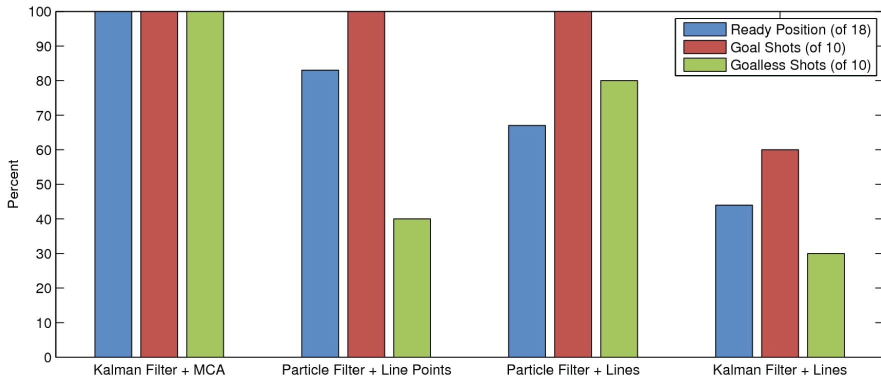


Fig. 8. Localization System Test Results

In the goal shot tests the success criteria was a lot simpler. The percentage represents the number of times the robot successfully kicked the ball towards the goal with an accurate position estimate. Kick line up issues were accounted for by monitoring the robot's position estimate throughout the tests. Successful goals where the robot's estimate was incorrect were not counted.

4.2 Computational Performance

The execution time of all four localization systems was monitored throughout the 3 tests. It should be noted that a small amount of debugging functions were enabled during the testing of all 4 systems and as a result execution times without any debugging functions may in fact be slightly lower. The execution time of the PF + Points algorithm is significantly longer than the 3 other algorithms

tested because although it lacks the optimisation in the full MCA, it requires the repeated projection of all line points for each of the 100 particles.

Algorithm	KF + MCA	PF + Points	PF + Lines	KF + Lines
Avg Time (ms)	6	10	4	3
Max Time (ms)	7.7	20	5	3

5 Conclusion

In this paper we have described a number of modifications and extensions to our original MCA implementation. Many aspects of the system have been looked at to address some of the issues and future work discussed in our previous paper. The results include: (i) revised and more computationally efficient Kalman Filter integration; (ii) usage of post information in the MCA to improve localization performance; (iii) MCA variance calculation to improve Kalman Filter interaction; (iv) point sampling for improved computational performance; (v) matrix method and chain multiplication optimisation for computational performance; and, (vi) efficient field marking point filtering for reduced false positive points.

Computational performance was previously a significant issue with the system but is clearly no longer a concern. The localization performance achieved with the enhancements listed above is superior to the previous version of the MCA and all other localization systems tested. As a testament to the system's high performance it was successfully demonstrated as the core part of RoboEireann's Open Challenge demonstration at RoboCup 2011, "Localisation without goal posts". The demonstration was voted 1st place out of 20 other presentations.

5.1 Future Work

Extensions to the proposed algorithm are needed to better deal with cases where the robot gets lost. This may occur due to a range of circumstances such as: (i) when the robot falls over; (ii) when the robot's locomotion is restricted, particularly when it is attempting to perform a rapid turn, but is blocked from being able to execute the turn; (iii) when the robot is moved by the game referees (for example in relation to either a local game stuck; or if the robot is penalized).

At present, the overall algorithm is slow to respond to these 'kidnapped robot' type situations, and further algorithm development is needed in this area. Multiple model Kalman filtering [6]; the ability to perform a larger number of MCA updates (with a larger number of sigma points) and the ability to perform multiple iterations of the local search may all improve the re-localization performance significantly. However, at least on the current hardware and with the current versions of the algorithm, speed improvements are crucial to permit these features.

References

1. Cox, I.: Blanche - An experiment in guidance and navigation of an autonomous robot vehicle. IEEE Transactions on Robotics and Automation 7(2), 193–204 (1991)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Section 15.2: Matrix-chain multiplication. In: Introduction to Algorithms, 2nd edn., pp. 331–339. MIT Press and McGraw-Hill (2001)
3. Julier, S., Uhlmann, J.: Unscented filtering and nonlinear estimation. Proceedings of the IEEE 92(3), 401–422 (2004)
4. Julier, S., Uhlmann, J.: Using covariance intersection for SLAM. Robotics and Autonomous Systems 55(1), 3–20 (2007)
5. Lauer, M., Lange, S., Riedmiller, M.: Calculating the Perfect Match: An Efficient and Accurate Approach for Robot Self-localization. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 142–153. Springer, Heidelberg (2006)
6. Quinlan, M.J., Middleton, R.H.: Multiple Model Kalman Filters: A Localization Technique for RoboCup Soccer. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS, vol. 5949, pp. 276–287. Springer, Heidelberg (2010)
7. Rath, C.: Self-localization of a biped robot in the RoboCup domain, Master's Thesis. Institute for Software Technology, Graz University of Technology (2010)
8. RoboCup Technical Committee. RoboCup Standard Platform League (Nao) Rule Book, `http://www.tzi.de/spl/pub/Website/Downloads/Rules2011.pdf`
9. Röfer, T., Laue, T., Thomas, D.: Particle-Filter-Based Self-localization Using Landmarks and Directed Lines. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 608–615. Springer, Heidelberg (2006)
10. Stüdli, S.: Kalman Filtering approach for Localisation in RobotSoccer, Master's Thesis. Hamilton Institute, NUI Maynooth & Institute for Control, Swiss Federal Institute of Technology (ETH), Zurich (2011)
11. Van Der Merwe, R., Wan, E.: The square-root unscented Kalman filter for state and parameter-estimation. In: IEEE International Conference on Acoustics Speech and Signal Processing, vol. 6, pp. 3461–3464. Citeseer (2001)
12. Whelan, T., Stüdli, S., McDonald, J., Middleton, R.H.: Line Point Registration: A Technique for Enhancing Robot Localization in a Soccer Environment. In: Röfer, T., Mayer, N.M., Savage, J., Saranlı, U. (eds.) RoboCup 2011. LNCS, vol. 7416, pp. 258–269. Springer, Heidelberg (2012)

A NUPlatform for Software on Articulated Mobile Robots

Jason Kulk and James S. Welsh

School of Electrical Engineering and Computer Science
University of Newcastle
Callaghan, Australia

Abstract. The development of software for robot systems is an involved process, that frequently results in a robot specific system. However, through careful design, frameworks that can be used on multiple operating systems and robot platforms can be created. This paper proposes such a framework.

The framework makes use of a blackboard and a class hierarchy to enable high–level software modules to be robot and operating system independent. The blackboard is used to standardise the transfer of information and allow for the high–level modules to adapt to changes in the robot hardware in real–time. The class hierarchy encapsulates the platform dependent aspects and provides a means of implementation sharing between different platforms. Furthermore, the hierarchy for the behaviour and motion modules simplify the addition of robot specific sections, and allow them to coexist.

The NUPlatform framework has been applied to six different platforms, including four physical robots and a simulator, and runs under several different operating system. The framework has also been successfully used in several research projects to implement vastly different robot behaviours.

1 Introduction

Software for robotic systems examines incoming sensor data to generate useful actions to be executed by a set of actuators. The development of the software is a long and expensive operation, with contributions from many developers, across a diverse range of fields. Given the size of software systems for robots, a solution which is portable, configurable and maintainable is desirable.

Robots have numerous sensors and actuators, each requiring a driver to communicate with the software system. A robot is made of a unique set of sensors and actuators, resulting in a unique set of drivers, and a unique set of inputs and outputs for the software system. Consequently, software written for a robot is often tied to that particular robot.

Furthermore, a robotic software system is often comprised of several distinct modules, for example, a vision module, a world modelling module, a behaviour module and a motion module. These modules may also be robot–dependant

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 31–45, 2012.
© Springer-Verlag Berlin Heidelberg 2012

if communication with the drivers is not done through a sufficient hardware abstraction layer.

This chain of dependancies in the transfer of information from the hardware to the high–level software, results in a robot–dependent system. The chain can be broken by inserting layers of abstraction that implement standard interfaces that are applicable to all robots. In particular, an abstraction layer is required between the software and the robot hardware, and a standard interface is required for inter–module communication.

The comparison of different versions of the same system module is a common task that can be accelerated if the modules are hot–swappable. That is, the module can be replaced without restarting the entire system. Furthermore, a single robot platform may be required to perform several unrelated tasks, consequently, the replacement of modules specific to that task at runtime is desirable. For instance, the replacement of a behaviour module designed to play soccer, with one designed to perform a human–robot interaction experiment.

In this paper we propose a software architecture that implements a hardware abstraction layer providing a logical robot [1] to the software system that is identical across robot platforms. The logical robot is presented to the software system as a blackboard [2], capable of storing a large variety of different sensor and actuator data. In addition to the data itself, the blackboard stores information regarding the data's validity, enabling software modules to detect and adapt to sensor or actuator faults.

The software architecture also makes use of class hierarchies to simplify the development of modules and drivers, and to enforce standard interfaces between submodules. In particular, the architecture is designed to keep the robot drivers quite thin, minimising the robot–platform specific implementation.

The annual RoboCup soccer competitions are a popular event where universities compete against each other in games of robot soccer [6]. The competition itself provides motivation for a large number of students and researchers, and results in an enormous body of software designed to win a soccer match. Unfortunately, the software written for RoboCup is typically not portable, or applicable to other research domains. Recently, there have been developments to correct this problem, predominantly from RoboCup teams who compete in multiple leagues.

The primary purpose of the software architecture proposed in this paper was to serve as the basis for the NUbot's entry into RoboCup's Standard Platform and Humanoid Leagues, which it has been successful in since 2009. However, the system was designed to accommodate similar legged robots and to enable the performance of a variety of tasks. In particular, the software architecture has facilitated the use of software developed for RoboCup to be used on other robot platforms, and to be used to perform tasks outside of the soccer domain.

1.1 Related Work

There are many software frameworks for robotic systems in the literature [3]. The method of transfer of information between hardware and system modules

can be categorised into two classes; those which are message–based and those which use a blackboard.

A message–based approach excels in distributed systems, as a message can be easily serialised and sent over a network. Notable examples of such systems are ROS [4] and YARP [5], both of which provide an excellent framework for robot independent software. However, the serialisation of the message and the data copying on either end, adds overhead to the system. The transport of the message through a physical network adds latency, however, on a single processor system shared memory can be used instead.

A blackboard is well suited to a single mobile processor system because of its lightweight. Blackboards are frequently used in the RoboCup legged soccer domains, given the limited processing available on small mobile robots. Both [7] and [8] use blackboards to share information between system modules.

The architecture proposed in this paper also uses a blackboard, as many of the target robots have limited processing power. The blackboard proposed in this paper is similar to that of [7], in that information is grouped into a small set of classes. The blackboard also incorporates an actuator command queue similar to that of NaoQi [9], enabling the storage of motion sequences and animations. However, in contrast, the blackboard used here has been generalised to encompass a wide variety of sensors and actuators, allowing the support for multiple robot platforms. In addition to the sensor information itself, the validity of the data is also stored in the blackboard, allowing higher level software modules to detect and adapt to sensor and actuator failure.

A hybrid blackboard–message based architecture is proposed in [10], to take advantage of the strengths of both approaches. A similar feature is used in this architecture, where commands, called jobs, can either be shared using the blackboard, or serialised and sent through a physical network. Although, the messages in this architecture are more aimed toward teleoperation, than distributed computing.

Few frameworks provide hardware drivers or implementations of common algorithms. Player [11] and ROS [4] are important exceptions, providing a large software base. Both the Player and ROS frameworks are component–based systems, where a robot is assembled from a set of existing software components. The alternative approach is to use inheritance to allow implementation sharing between similar robot platforms. In this architecture we use a structured class hierarchy to minimise the implementation of drivers and modules, the class hierarchy also enforces standard interfaces. This approach is conceptually similar to several software frameworks for robots, in particular, RoboFrame [12].

Given the dominant target robot platforms for this architecture are legged robots, the configuration of the motion system is important. It is common to use a motion manager to select from a list of available motion providers which provider is going to control the robot [13, 14, 7]. This architecture uses the same general principle, however, each limb can be controlled by a separate provider which, for example, frees the arms of a biped to perform other tasks while walking. Additionally, a bridge pattern [17] is used to separate the motion manager

from the walk engine, allowing the implementation of vastly different engines, in particular engines for both biped and quadruped robots.

Finally, of all the robot software frameworks reviewed in this section, none of them have support for the niche set of target robots. This was an important consideration, and resulted in the development of the software architecture which is presented in the next section.

1.2 Overview

Figure 1 outlines the NUPlatform software architecture. The key parts being the blackboard, the system modules, and the hardware drivers. The NUPlatform itself is open source, and publicly available at [15], and written entirely in C++.

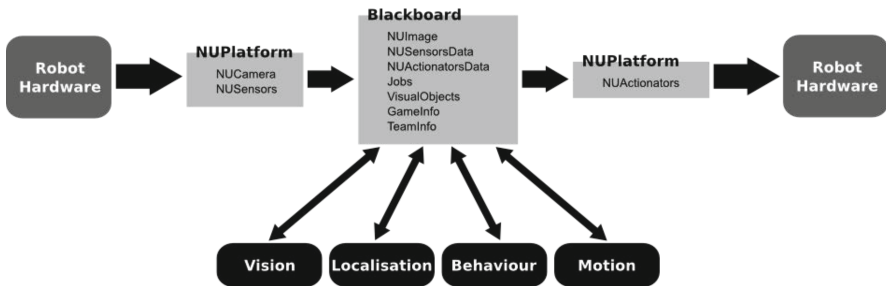


Fig. 1. An overview of the software architecture, and the transfer of information between the hardware and software modules via the blackboard.

The blackboard is central to the system, all of the information transferred between modules is done via the blackboard. The NUPlatform object itself, populates the blackboard with sensor data received from hardware, and gives the commands stored in the blackboard to the hardware.

The remaining system modules; vision, localisation, behaviour and motion, also communicate using the blackboard. Sensor data is obtained from the blackboard, and the results of the execution of each module are then stored on the blackboard for the other modules to use.

2 The Blackboard

The purpose of a blackboard in a software system is to store information, and share that information with any modules that require it. Given that the blackboard can be updated and accessed from many threads, it needs to be thread–safe. The blackboard is also used from within real–time threads, so it needs to be very efficient. Furthermore, as it is an object that all developers will be using frequently, it should be easy to use.

The blackboard used as part of the NUPlatform is shown in Fig. 2. The information stored in the blackboard is grouped into seven classes based on the source of the information. Each of the classes will be discussed below.

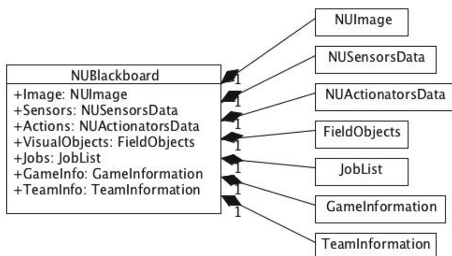


Fig. 2. A UML class diagram of the blackboard showing the seven constituent parts

2.1 Sensors

The purpose of the `NUSensorsData` is to store the sensor data produced by the robotic hardware so that it can be accessed by the rest of the software system. A UML class diagram of the sensor data store is shown in Fig. 3.

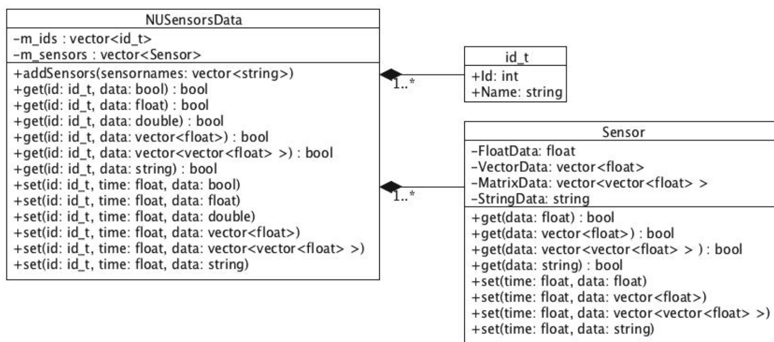


Fig. 3. A UML class diagram of the NUSensorsData, the class which stores all of the sensor data on the blackboard

The data for an individual sensor is initialised as being invalid. When the data is updated using the `set` function it becomes valid. Consequently, for a sensor which is never updated, which is the case for a sensor that is not present on a particular robot platform, the data will remain invalid and the `get` function will always indicate to the rest of the software that the sensor is not available.

Furthermore, it is possible for sensor data to be invalid even on a robot platform that has the particular sensor. The data may become invalid because of a

hardware fault, such as the loss of communication with a single sensor, or by design, such as the kinematically calculated camera height for a humanoid robot becoming invalid when the robot is no longer on the ground.

2.2 Actuators

The purpose of the `NUActionatorsData` is to store commands produced by the software system to be given to the robotic hardware. The command store shares several properties with the sensor data store, and in fact inherits from the same base class. A UML class diagram is shown in Fig. 4.

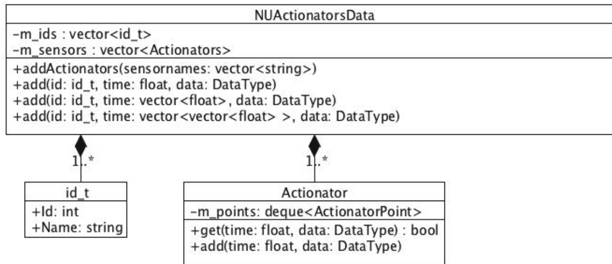


Fig. 4. A UML class diagram of the NUActionatorsData, the class which holds all of the actions to be given to the robotic platform

Conceptually, a command consists of data describing the action, and a timestamp at which the action should be performed. This is encapsulated by the `ActionatorPoint` object; storing data of several common types accepted by hardware actuators, and the timestamp at which the action should be executed.

To simplify the higher–level software modules the `Actionator` object is used to store a queue of `ActionatorPoint` objects. The queue is sorted based on the timestamp for each command, and interpolation is done by default on numeric data between consecutive commands. Being able to queue commands makes implementing sequences of actions straightforward, for example, a motion script for a joint or an animation for an LED can be stored in an `Actionator`.

Fig. 4 shows that the interface to the command store consists of a single `add` function. To understand how the interface works, consider the following function signature as an example

```
add(id_t id, vector<double> time, vector<float> data).
```

There are several ways a user might wish to use such a function. If the `id_t` addresses a single actuator, then the timestamps and command data will be formatted as

$$[time_0, time_1, \ldots, time_N][data_0, data_1, \ldots, data_N].$$

In this instance, it is clear the user is specifying a sequence of commands to a single actuator.

However, if the `id_t` addresses a group of actuators the intention of the user is ambiguous. The timestamps and command data will be formatted as

$$[time_0, time_1, \ldots, time_M][data_0, data_1, \ldots, data_L].$$

If M and L both match the number of actuators in the group, the user is applying a single command to each actuator with a unique timestamp. However, if M and L do not match, then the user is applying the same vector with a different timestamp to each actuator in the group.

By applying logic to the interpretation of commands specified by the user, the ambiguity in the command can be removed. A similar approach to the one described in the example is applied to each of the `add` functions. In the event that a command does not match any of the possible formats, the command is discarded, and the user is alerted that their command was incorrectly formatted.

2.3 Visual Information

The visual information is stored in the blackboard in two objects; the `NUImage` and the `FieldObjects`. The `NUImage` stores the image data as YUV422 from the robot's vision sensors. Relevant settings used by the vision sensors at the time the images were captured, such as the resolution, exposure and hue, are also stored in this object.

The `FieldObjects` stores the visual information extracted from the images after object detection. Each detected object stored in the `FieldObjects` contains its relative position and velocity from the robot. This object serves as the primary source of information for the world modelling to be performed.

2.4 Jobs

The purpose of the `JobList` in the blackboard is to store jobs to be executed by a software module. This is distinct from the `NUActionatorsData`; jobs in the `JobList` encapsulate a task at a much higher level, and are to be executed by software modules, not by hardware.

Fig. 5 shows a UML class diagram of a subset of the available jobs. A class hierarchy is used to share implementation among similar jobs. An STL iterator is implemented so that each software module can iterate over the jobs in the list, and execute the jobs assigned to it.

For example, consider the `WalkJob`, which controls the movement of the robot. A `WalkJob` is typically generated by the behaviour module. However, a `WalkJob` can be generated on an external system, and sent to a robot via a network. In effect this enables the robot to be very easily remote controlled, whether by a human operator, or by another artificial agent. Furthermore, every robot is also capable of transmitting a `WalkJob` over a network, thus enabling any robot to control any other robot.

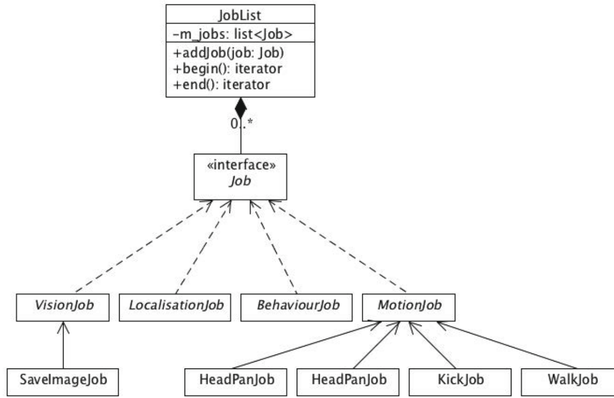


Fig. 5. A UML class diagram of the `JobList`, showing a subset of the class hierarchy that makes up the available jobs

2.5 Network Information

Information received from the network is stored in the blackboard in two objects; `GameInformation` and `TeamInformation`. The origin of the class names stem from RoboCup soccer; the `GameInformation` encapsulating the state of the soccer game [16], and the `TeamInformation` encapsulating the state of each of a robot's team mates. However, these concepts can be used in more general domains.

The *game* need not be a soccer match, it could be any sort of task. We use the referee for RoboCup to allow a human supervisor to control the basic behaviour of the robot, such as starting, stopping and pausing the robot.

The *team* can consist of robots performing any task. The `TeamInformation` stores the last known position of each robot in the team, as well as the task it was executing.

3 The Platform

Fig. 6 shows the organisation of the robot dependent module of the software system. The platform consists of four modules; the `NUPlatform`, `NUCamera`, `NUSensors` and `NUActionators`, each of the components will be discussed below.

The flow of information through the four modules is shown in Fig. 7. The `NUCamera` and `NUSensors` encapsulate the input sensors of the robot, while the `NUActionators` encapsulates all the actuators. These three objects isolate the high–level software modules from the robot hardware. Furthermore, the `NUPlatform` encapsulates the underlying operating system. Consequently, the high–level software modules can be made both robot and operating system independent.

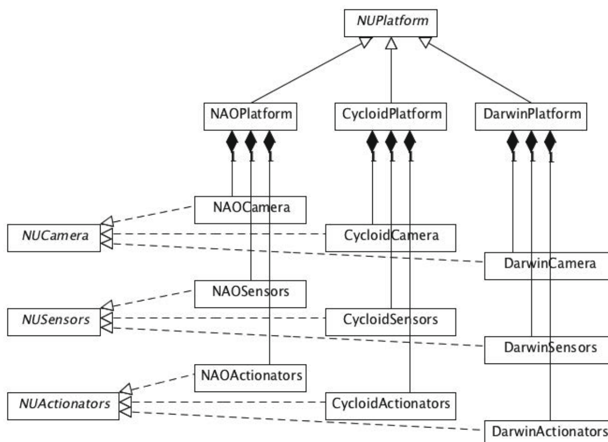


Fig. 6. A UML class diagram of the NUPlatform, showing how the class hierarchy is organised with three example robot targets

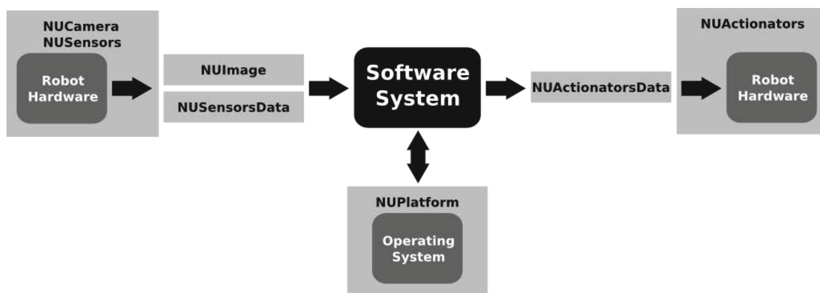


Fig. 7. An overview of the transfer of data between the software modules and hardware using the NUPlatform framework

3.1 NUPlatform

The `NUPlatform` provides a robot and operating system independent interface for system calls. This includes functions to access the time, threading, and networking of the underlying operating system, and also provides functions regarding a robot's identity. The class also houses the robot dependent camera, sensor and actuator modules.

3.2 NUCamera

The `NUCamera` provides an interface to the robot's vision sensor. The `NUCamera` has the simple purpose of copying the raw image to the `NUImage` on the black-

board. A simple interface to modify the camera settings is also provided by this class.

The implementation of the class itself is robot dependent. The implementation may be inherited from a generic `NUOpenCVCamera` or `NUV4LCamera`, or it may be robot specific, as is the case with the `NAOCamera`.

3.3 NUSensors

The primary role of the `NUSensors` module is to copy data produced by hardware sensors into the `NUSensorsData` on the blackboard. Before copying the data the `NUSensors` converts the data into the appropriate format to be stored on the blackboard. This includes the reduction of the data down to one of the accepted types, but also includes necessary scaling and ordering to ensure unit and sign conventions are preserved.

The secondary role of the `NUSensors` is to calculate soft sensors and select the best sensor readings to provide a particular sense. For example, consider the orientation of the torso of a humanoid robot. The orientation may be provided by an IMU, in which case this sensor is used. However, the hardware may only have accelerometers and gyrometers, in which case the orientation needs to be calculated. Furthermore, the orientation of the torso can be calculated using the kinematic chain of the supporting leg. When both accelerometers and kinematics are valid, a Kalman filter is used to fuse the information together, however, when there is a sensor fault with accelerometers, or the robot is not on the ground only the single valid sensor readings are used.

3.4 NUActionators

The purpose of the `NUActionators` is to copy the commands, stored in the `NUActionatorsData` on the blackboard, to the robot's actuators. Like the `NUSensors`, the data stored on the blackboard needs to be converted into the proper format expected by the hardware. This process, along with the actual data transfer, is robot dependent.

4 Software Modules

4.1 Behaviour

The goal of this software module is to provide task orientated behaviour for a robot. The behaviour required of a robot is very specific to a target application, and the target applications are vastly different. To provide behaviour for a wide variety of tasks the system outlined in Fig. 8 is used.

The system has a single `Behaviour` class which serves as a manager, allowing the selection of a `BehaviourProvider` to implement the task orientated behaviour. The selection of an appropriate behaviour can be done at compile–time, by specifying a default behaviour, or by using a button interface while the

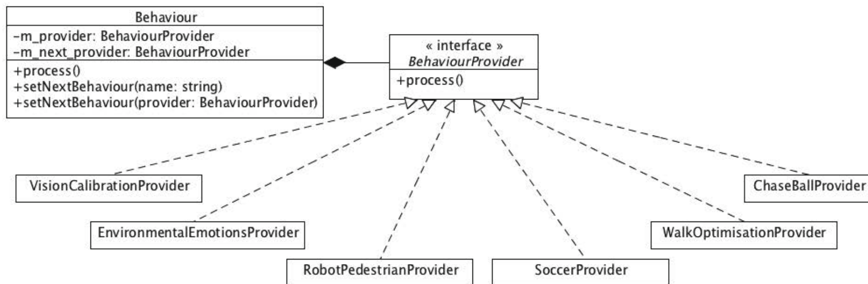


Fig. 8. A UML class diagram of the Behaviour system, showing a subset of the available behaviour providers

robot is running. The latter approach is extremely useful when using robots in the field.

The online behaviour switching is done at the beginning of a behaviour cycle. The `Behaviour` class checks if another behaviour has been requested. If so, then the new behaviour is created, set as the current behaviour, and then the previous behaviour is terminated.

4.2 Motion

The motion system provides the robot with a means to move around in its environment. Fig. 9 shows an overview of the modules that make up the motion system. The general principle behind the system is to have a set of motion providers capable of controlling the robot, and use a motion manager to select which of the providers should be running at any given time.

The `NUMotion` class is the motion manager, selecting a motion provider for the head, arms and legs. The manager decides which provider to execute based on information in the `NUSensorsData` and the `JobList`. The sensor data is used to determine when the `FallProtection` and `Getup` providers should interrupt the current motion providers. The jobs provide a much smoother transition between providers, waiting for one provider to finish before starting the next.

The motion modules need to be suitable for each of the target robot platforms. The `NUHead` provider controls the motion of the head in a robot independent manner, providing an interface to perform common tasks such as panning and tracking an object. In the case of the `Script`, `Getup`, and `FallProtection` providers, robot dependent configuration files are used to tailor the motions to a specific robot.

The `NUWalk` provider is the most difficult provider to port to each robot platform given the vastly different methods of robot locomotion. A bridge pattern [17] is used to separate the motion manager from the walk engine implementations. It is desirable to use the same walk engine on different robots [13], however, this is not always possible. A walk engine may only run on single robot,

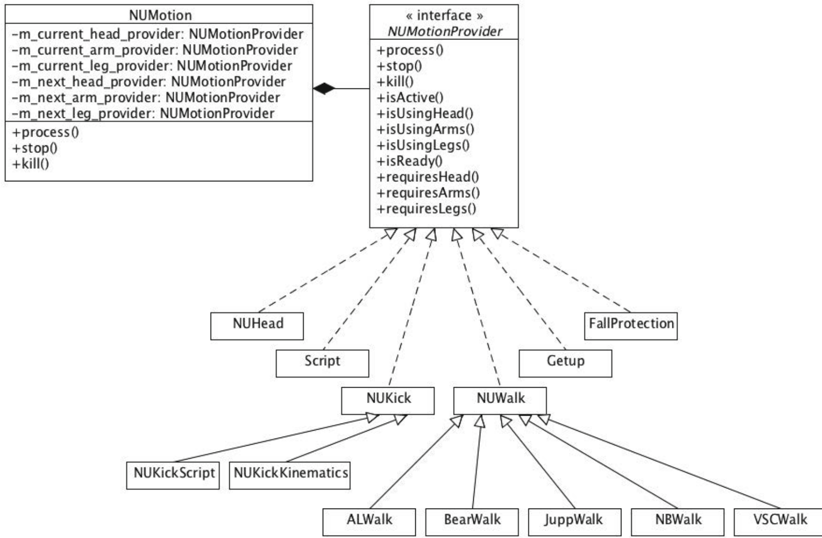


Fig. 9. A UML class diagram of the Motion system, showing the hierarchy of motion providers

such as Aldebaran Robotics' walk engine for the NAO, or the robots are too dissimilar, such is the case between wheeled and legged robots.

The `NUWalk` provider selects the appropriate engine to be used with each platform. In the case that multiple engines can be used, the selection is left to the user, and robot specific walk parameters are used to tailor an engine to a particular robot.

5 System Configuration

The software system is configured using CMake [18]. Each robot platform has a configuration file specifying the required external libraries and platform dependent source files, as well as default values for miscellaneous configuration variables. The result of which is the system can be built for a particular target using simple commands like `make NAO` or `make Cycloid`.

The user is also able to configure many aspects of the build. In particular, the user can select which system modules to include in the build, that is the user is able to select whether vision, localisation, behaviour and motion should be compiled. In the instance where multiple implementations of the same module are provided, the user is able to select which implementation to use. For example, the particular walk engine to compile can be selected.

6 Applications of NUPlatform

There are currently six platforms supported by the NUPlatform software architecture. This includes four physical robots shown in Fig. 10, three bipeds and one quadruped. In addition to these robots the framework also supports the Webots simulation package [19], and a generic Webcam. The amount of platform specific code is quite small, at approximately 500 lines per supported platform.

The Webots and Webcam platforms have also been successfully used under the Linux, Mac OS-X and Windows operating systems. However, all of the physical robots run Linux.

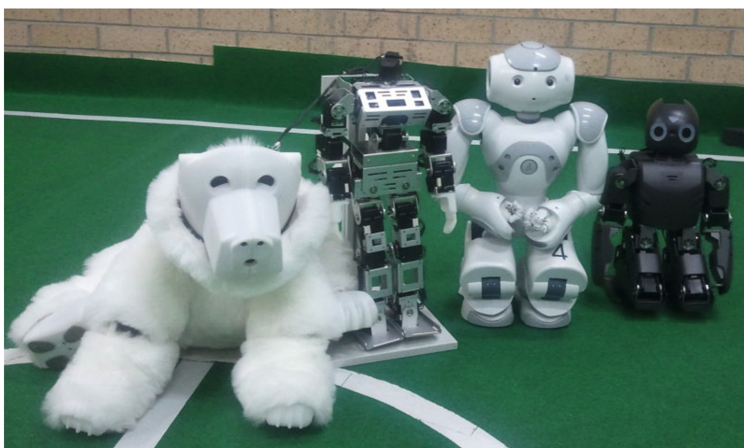


Fig. 10. The physical robots currently running the NUPlatform. From left to right; the HyKim [20], a modified CycloidII [21], the NAO [9] and the DARwIn [22].

The software framework has also be used in a range of projects. The first major project is the Standard Platform League at RoboCup [23], and related projects [24, 25], where the software framework proposed in this paper has been used since 2009. The other major project to use the framework was the design of urban spaces through pedestrian analysis [26], where robot pedestrians were used as an intermediate step between simulation and real–world experiments.

7 Conclusion

The NUPlatform software architecture provides a robot and operating system independent framework for the development of robot software. In particular, a robot independent method for the transfer of information from hardware to high–level modules is provided.

A structured class hierarchy is used to minimise the amount of robot dependent implementation. The class hierarchy also allows the interchange of different implementations of the same modules.

The NUPlatform's flexibility has been demonstrated through its application to six different platforms, including four different physical robots. The framework has also been used in several different projects to provide vastly different robot behaviours.

References

1. Alami, R., Chatila, R., Fleury, S., Ghallab, M., Ingrand, F.: An architecture for autonomy. The Int. Journal of Robotics Research 17(4), 315 (1998)
2. Jagannathan, V., Dodhiawala, R., Baum, L.: Blackboard architectures and applications, vol. 3. Academic Press, Boston (1989)
3. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: A survey. Autonomous Robots 22, 101–132 (2007), `http://dx.doi.org/10.1007/s10514-006-9013-8`, doi:10.1007/s10514-006-9013-8
4. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
5. Fitzpatrick, P., Metta, G., Natale, L.: Towards long-lived robot genes. Robotics and Autonomous Systems 56(1), 29–45 (2008)
6. Robocup standard platform league, `http://www.tzi.de/spl/`
7. Röfer, T., Laue, T., Müller, J., Burchardt, A., Damrose, E., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T., et al.: B-human team report and code release 2010. Technical report (2010), `http://www.b-human.de/en/publications`
8. Barrett, S., Genter, K., Hausknecht, M., Hester, T., Khandelwal, P., Lee, J., Quinlan, M., Tian, A., Stone, P., Sridharan, M.: Austin villa 2010 standard platform team report. Technical Report UT-AI-TR-11-01, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, Tech. Rep. (2011)
9. Aldebaran robotics' nao humanoid robot, `http://www.aldebaran-robotics.com/`
10. Niemueller, T., Ferrein, A., Beck, D., Lakemeyer, G.: Design Principles of the Component-Based Robot Software Framework Fawkes. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 300–311. Springer, Heidelberg (2010)
11. Collett, T., MacDonald, B., Gerkey, B.: Player 2.0: Toward a practical robot programming framework. In: Proc. of the Australasian Conf. on Robotics and Automation, ACRA 2005 (2005)
12. Petters, S., Thomas, D., Von Stryk, O.: Roboframe-a modular software framework for lightweight autonomous robots. In: Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (2007)
13. McGill, S., Brindza, J., Yi, S., Lee, D.: Unified humanoid robotics software platform. In: 5th Workshop on Humanoid Soccer Robots (2010)
14. Northern bites' robocup code repository (2010), `http://github.com/northern-bites/nao-man`
15. Kulk, J., Nicklin, S., Wong, A., Bhatia, S.: Nubot's robocup code repository (2011), `http://github.com/nubot/robocup`
16. Robocup gamecontroller, `http://sourceforge.net/projects/robocupgc/`

17. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
18. Kitware. Cmake: Cross platform make
19. Cyberbotics webots, http://www.cyberbotics.com/products/webots/
20. Tribotix: Hykim (robot bear), http://www.tribotix.com/Products/Tribotix/Robots/Hykim_info1.htm
21. Robotis cycloidii humanoid, http://www.tribotix.com/Products/Robotis/Humanoids/CycloidII_info1.htm
22. Robotis darwin humaniod robot, sourceforge.net/projects/darwinop/
23. Nicklin, S.P., Bhatia, S., Budden, D., King, R.A., Kulk, J., Walker, J., Wong, A.S., Chalup, S.K.: The nubots' team description for 2011, University of Newcastle, Tech. Rep (2011), http://www.tzi.de/spl/pub/Website/Teams2011/NUbotsTDP2011.pdf
24. Nicklin, S., Welsh, J.S.: Forward kinematic based biped odometry. University of Newcastle, Tech. Rep. (2011)
25. Kulk, J., Welsh, J.S.: Using redundant fitness functions to improve optimisers for humanoid robot walking. In: International Conference on Humanoid Robotics (2011)
26. Wong, A.S.W., Chalup, S.K., Bhatia, S., Jalalian, A., Kulk, J., Ostwald, M.J.: Humanoid robots for modelling and analysing visual gaze dynamics of pedestrians moving in urban space. In: The 45th Annual Conf. of the Australian and New Zealand Architectural Science Association, ANZASC (2011)

Service Component Architectures in Robotics: The SCA-Orocos Integration

Davide Brugali[1], Luca Gherardi[1],
Markus Klotzbücher[2], and Herman Bruyninckx[2]

[1] University of Bergamo, DIIMM, Italy
{brugali,luca.gherardi}@unibg.it
[2] Katholieke Universiteit Leuven, PMA, Belgium
{markus.klotzbuecher,herman.bruyninckx}@mech.kuleuven.be

Abstract. Recent robotics studies are investigating how robots can exploit the World Wide Web in order to offer their functionality and retrieve information that is useful for completing their tasks. This new trend requires the ability of integrating robotics and information systems technology. On the first side a set of robotics component based frameworks, which are typically data flow oriented, have been developed throughout the last years and Orocos is one of the most mature. On the other side the state of the art is represented by the Service Oriented Architecture, where the Service Component Architecture defines a component-based implementation of this approach.

The paper reports the progress of our work, which aims to promote in the robotics field a cooperation between Service Oriented Architecture and Data Flow Oriented Architecture. To achieve this we propose an integration between SCA and Orocos. We first highlight a set of architectural mismatches that have to be faced for allowing this integration and then we introduce a java-based library, called JOrocos, that represents our solution to these mismatches. Finally we describe a case study in which SCA and Orocos components cooperate for monitoring the status of a robot.

1 Introduction

Recent advances in robotics and mechatronic technologies have stimulated expectations for emergence of a new generation of autonomous robotic devices that interact and cooperate with people in ordinary human environments.

Engineering the control system of autonomous robots with such capabilities demands for technologies that allow the robot to collect information about the human environment, to discover available resources (physical and virtual), and to optimally exploit information and resources in order to interact with people adequately. Common approaches in robotics build on sophisticated techniques for perception and learning, which require accurate calibration and extensive off-line training

Recent approaches investigate how the robot can exploit the World Wide Web to retrieve useful information such as 3D models of furniture [10] and images of

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 46–60, 2012.
© Springer-Verlag Berlin Heidelberg 2012

objects commonly available at home [15]. In [16] the *Robotic Information Home Appliance* is illustrated as a home robot interconnected to the home network that offers a friendly interface to information equipment and home appliances.

This new trend poses new challenges in the development of robot software applications since they have to integrate robotic and information systems technologies, which account for quite different non-functional requirements, namely performance and real-time guarantees at one side and scalability, portability, and flexibility at the other side.

Modern robot control systems are typically designed as (logically) distributed component-based systems, where the interactions between components (control, sensing, actuating devices) are usually more complex compared to more traditional business applications. In Robotics, the software developer faces the complexity of event-based and reactive interactions between sensors and motors and between several processing algorithms. For this reason, robotic-specific component-based models and toolkits have been developed, which offer mechanisms for real-time execution, synchronous and asynchronous communication, data flow and control flow management, and system configuration.

In contrast, the most common middleware infrastructures for the World Wide Web and home networks are the Java Platform Enterprise Edition and Service Oriented Architectures. Service Oriented Architectures (SOA) have been proposed as an architectural concept in which all functions, or services, are defined using a description language and where their interfaces are discoverable over a network [8].

Some attempts to develop robotic applications as SOA systems can be found in the literature (a recent survey can be found in [5]). Their main disadvantage is that they give up the typical component-based nature of robotics systems and force a pure service oriented approach.

More recently, Service Component Architectures (SCA) [12] have been proposed as an architectural concept for the creation of applications that are built by assembling loosely coupled and interoperable components, whose interactions are defined in terms of bindings between provided and required services. As such, SCA offer the advantages of both the Component-based engineering approach typically used in robotics and the Service Oriented Architectures.

In order to bridge the gap between current component-based approaches to robotic system development and modern information systems technologies, we have developed the JOrocos library that extends the popular Orocos robotic framework [11] with Java technologies. Thanks to the JOrocos library, a robot control application can be designed as a SCA system, where components encapsulating real-time control functionality are seamlessly integrated with web services and the most common Java toolkits, such as the SWING framework for developing graphical user interfaces.

The paper is organized as follows. Section 2 introduces the Service Oriented Architecture and its main features. Section 3 briefly presents the Orocos concepts that are useful for better understanding the paper. Section 4 describes the architectural mismatches between SCA and Orocos and how JOrocos is designed in

order to define a bridge between the two component models. Section 5 presents a simple case study in which JOrocos is used for implementing an application where SCA and Orocos components work together. Finally section 6 draws the relevant conclusion.

2 Service Component Architecture

Robot control applications are increasingly being developed as component-based systems [4]. The reason is that, ideally, components embedding common robot functionality should be reusable in different robot control systems and application scenarios

This is achieved by clearly separating the component specification, which should be stable [3], from its various implementations.

A component specification explicitly declares which functionality (provided interfaces) are offered to its clients (code invoking an operation on some component, not necessarily in a distributed environment), and the dependencies (required interfaces) to the functionality that are delegated to other components.

Separating component specification from its implementation is desirable for achieving modular, interoperable, and extensible software and to allow independent evolution of client and provider components. If client code depends only on the interfaces to a component and not on the components implementation, a different implementation can be substituted without affecting the client code. Furthermore, the client code continues to work correctly if the component implementation is upgraded to support an extended specification [4].

The Service Component Architecture defines a generalized notion of a component, where provided interfaces are called *Services* and required interfaces are called *References*. Services and references are thus typed by interfaces, which describe sets of related operations that can be invoked synchronously or asynchronously.

The communication between pairs of components (i.e. operation invocation) occurs according to the specific binding protocol associated to *services* and *references*. A single service or reference can have multiple bindings, allowing different remote software to communicate with it in different ways, i.e. the WSDL binding to consume/expose web services, the JMS binding to receive/send Java Message Service, the Java RMI binding for classical caller/provider interactions.

The components in a SCA application might be built with Java or other languages, or they might be built using other technologies, such as the Abstract State Machines Language (ASML) [14][2]. Components can be combined into larger structures called composites [13], that are to be deployed together and that can themselves be further combined. Components in a composite might run in the same process, in different processes on a single machine, or in different processes on different machines.

The structure of SCA components and their interconnections are defined using an XML-based metadata language (SCDL), by which the designer specifies: the set of *services* provided and the *references* required by each component; the implementation of each service as a link to a Java or other programming languages

file; the component *properties*, i.e. data for configuring the component functionality, whose values are not hard-coded in the component implementation; the associations between references and services of different components; the bindings that specify access mechanisms used by services and references according to some technology/protocol; the aggregation of components in *Composites*.

SCA is supported by graphical tools and runtime environments. The tools build on the Eclipse Modeling Framework and allow the generation of a SCDL configuration file from a graphical representation of components and systems. SCA runtime environments, like Apache Tuscany and FRAscaTI, parse the configuration file, instantiate the implementation of each component, assign values to component properties, locate component services and references and create bindings with pairs of services and references.

3 The Orocos Framework

Orocos is one of the oldest open source framework in robotics, under development since 2001, and with professional industrial applications and products using it since about 2005. The focus of Orocos has always been to provide a hard real-time capable component framework — the so-called *Real-Time Toolkit* (RTT) implemented in C++ — and as independent as possible from any communication middleware and operating system.

In the context of this paper, the following Orocos primitives are most relevant for the development of real-time components[1]:

- *TaskContext*: this is the Orocos version of a component, providing the basic infrastructure to make a system out of pieces of code that can interact with each other via data and events, a default *life cycle state machine*, reporting and timing support, an operating system abstraction layer, etc. The latter is, for example, provided via the primitive of an *Activity*, that is being mapped onto the process or threading system of the underlying operating system.
- *Data Ports*: components provide or request certain data types on their interface, in order to allow data flow based application architectures. Such architectures are very common in real-time control applications, since those typically perform the same kind of operations and interactions all over again, triggered by time or by hardware events. Hence, data flow is used for implementing the business logic of real-time components as it allows the developers to guarantee that a set of operations will be executed in a fixed amount of time: the period of the component. Figure 1 shows a system of Orocos components connected through their data ports according to the data-flow paradigm. It is taken from the Component Builder's Manual[1].
- *Connection policy*: while Orocos wants to be as independent as possible from communication middleware (in order to let users make their choice

[1] The technical details can be found on the project's website, and more in particular in the *The Orocos Component Builder's Manual* `http://www.orocos.org/wiki/orocos/toolchain/toolchain-reference-manuals`.

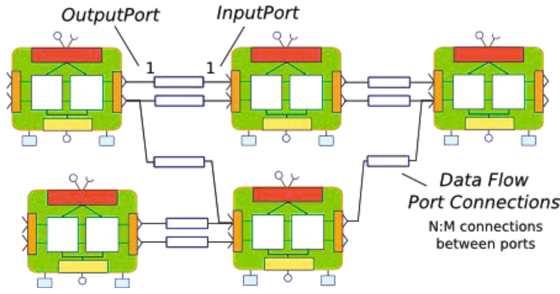


Fig. 1. The Orocos Data Flow Oriented Architecture

based on their own criteria), it does provide some “middleware” anyway, for those interactions that take place in-process (for hard real-time data exchange between threads). Ports hide the low-level locking primitives of the underlying operating system, so that users don’t have to worry about them, but the design goal of being able to guarantee real-time performance is not well served by a *complete* shielding of the way *Activities* interact. Hence, Orocos provides *Lock free data Ports* as one of its major features: an *Activity* will never have to wait to get or read data when it needs them, because Ports can be configured to copy their data when one or more other Activities are also accessing the data.

– *Services*: Orocos implements the concept of services, which are containers of operations. They are not used for implementing the kind of operations that regard the business logic of the real-time components, but for example for configuring their period or retrieving information about their status (stopped, running, etc).
– *Properties* are part of the Service Configuration interface and are used to load or tune application specific configurations at runtime (e.g. the parameters of a PID).

4 SCA - Orocos Integration

In order to make possible the interaction between SCA and Orocos components we had to face some architectural mismatches presented by the two frameworks. In fact, despite both SCA and Orocos components interact by exchanging messages, the syntax and semantics of these messages is fundamentally different.

In SCA messages are used for invoking services provided by components. Services are defined by explicit interfaces that completely describe the name of each operation, its arguments and the return value (the signature of the method). The message sent by the requester component to the provider component describes which operation has to be executed and provides its parameters. Hence the execution of the component functionality starts when the message is received.

In Orocos instead the communications are based on data flows and the messages are used for exchanging data. Components periodically elaborate data received on

the input ports and write their results on the output ports. This means that the components business logic is regularly executed every T milliseconds, where T is the period of the component.

This meaningful difference introduces two main problems:

1. How an invocation of a SCA service can produce an input that will be processed in the next cycle of an Orocos component business logic?
2. How the data published on an Orocos output port can trigger the execution of a SCA service?

Let's introduce how we solved these problems by means of a simple scenario in which two SCA components and an Orocos component cooperate in order to move a Kuka youBot [9] towards a given position. The youBot is a mobile manipulator with an omnidirectional and holonomic base and a five degrees of freedom arm. The components are described below:

– A SCA component, called *Locomotor*, which provides a service for moving a youBot towards a position defined by the client and monitoring its activity. The component is in charge of transforming the given cartesian position in a set of commands (joint positions), forwarding them to the robot and retrieving its status. In order to do that the component requires two services, which are provided by the driver of the robot for sending and receiving these information.
– An Orocos component, called *youBotDriver*, which provides an input port and an output port. The component implements the API of the youBot and is in charge of actuating the axes in order to reach the joints positions specified by the client on the input port. The output port is instead used for periodically publishing the status of the robot, for example the position and the velocity of the joints.
– A SCA component, called *SCAyouBotDriver*, which is implemented by using the JOrcos library and represents a proxy to the youBotDriver component within the SCA system.

The *SCAyouBotDriver* is described by means of the following interfaces:

– Provided interface *sendingCommand*. This interface provides a service for receiving commands from the *Locomotor* and writing these commands on the input port of the *youBot Driver*. The result is the activation of the *youBot Driver* operations that are in charge of moving the robot.
– Provided interface *retrievingStatus*. This interface is invoked by the *Locomotor*. The *SCA youBot Driver* periodically checks and retrieves the new data available on the *youBot Driver* output port. The interface provides the operation for retrieving these values.
– Required interface *notifying*. This interface is provided by the *Locomotor* and is used by the *SCA youBot Driver* for notifying some events. The possible events are *idle*, *busy*, *refresh*. The component raises the event *busy* when it starts the execution of an operation and the event *idle* when this operation

is completed. In this way the *Locomotor* knows whether the operation that it requested is completed or not. The state *refresh* is instead raised when new data are read from the *youBot Driver* output port.

Here is important to consider that in order to notify the *Locomotor* about the availability of new data before the *youBot Driver* deadline, the *SCA youBot Driver* should check the output port with a frequency at least two times greater than the one of the Orocos component.

The components and the interfaces of this scenario are depicted in figure 2.

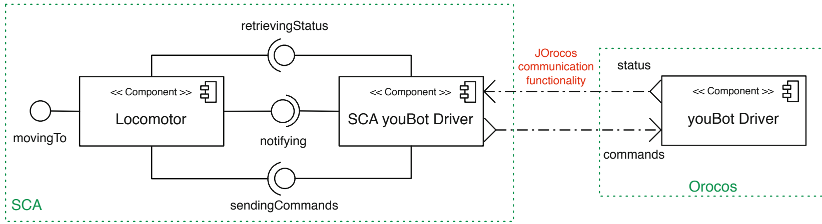


Fig. 2. The youBot Scenario

Another difference between SCA and Orocos regards the synchronization of the component operations after the action of sending a message. In SCA it is possible to define by means of an annotation whether the message is sent in a synchronous or asynchronous way. In the first case the thread that sends the message suspend itself until the result is returned. In the second case instead the thread continues its execution without waiting for the return value. A callback message will notify the component when the return value of the sent message will be computed. In Orocos all the messages are sent in an asynchronous way. The components read the data on the input ports and publish data on the output ports without waiting for other component activities.

Our library face this problem by providing the possibility of reading data from the Orocos output port in an asynchronous way, according to the Publish/Subscribe communication paradigm [6] (more information about the implementation will be described in the subsection 4.2). In this way both SCA and Orocos component don't have to wait after sending a message. However a synchronous communication can always be defined in the implementation of *SCA youBot Driver*.

The last mechanism provided by SCA that is not defined in Orocos is the hierarchical composition of the components. In SCA this functionality is available by using the concept of composite. A composite contains different components and allows the developer to promote a set of their services in order to make them accessible to the clients of the composite. In this way a composite can be reused as a simple component in a more complex architecture.

Here is possible to leverage on this SCA mechanism and create composites that contains different bridges to Orocos components (like the *SCA youBot Driver*) and promotes their operations as services. This approach is inspired by the facade design pattern, which aims to provide a unified interface to a set of interfaces in

a subsystem [7]. In this way a single reusable SCA component, the composite, can provide the functionality defined in several Orocos components to its client.

4.1 The JOrocos Library and Its Architecture

The JOrocos library offers a set of mechanisms that allow the implementation of the proxies of Orocos components mentioned in the previous pages (e.g. *SCA youBot Driver*). These mechanisms provide the functionality for reading and writing on Orocos data ports, reading and writing Orocos properties and invoking operations provided by Orocos components.

Another interesting mechanism offered by JOrocos is the introspection of Orocos running components. It provides the functionality for discovering at runtime which components are available, their ports, their operations and their properties. This mechanism allows the development of systems more complex than the scenario defined in introduction of this section: systems in which the SCA composite doesn't have a priori knowledge of the Orocos components and configures itself at runtime according to the information retrieved through the introspection. For example, with reference to the previous scenario, it will be possible to design a system in which the SCA composite doesn't know at compile time which robot has to be controlled. This information will be retrieved at runtime by introspecting the current *Robot Driver* component and according to its ports the *SCA Robot Driver* will configure itself.

This functionality is realized on the top of Corba, the middleware that Orocos uses for exchanging messages between distributed components. Corba doesn't guarantee the respect of real-time constraints and for this reason when the communication between Orocos components has to be real-time the components have to run on the same machine. In this way the communication between the local components doesn't rely on Corba and so the respect of the real-time constraints is not compromised. In the same way the use of Corba is not a problem for our integration because the real-time components will be implemented by using Orocos and will run on the same machine.

The architecture of the library is depicted in the UML class diagram reported in figure 3. As showed in the diagram the classes of the library are organized in two main packages: *core* and *corba*.

- The core package contains the classes that store data structures and offer operations that are middleware independent. These classes define the core of the library and represent the main entities of an Orocos system.
- The corba package contains instead the classes whose methods provide a set of operations that are corba specific.

The classes of the core package whose name starts with the word *Abstract* are abstract classes and have to be extended in order to provide the functionality that are middleware specific. They represent proxies of Orocos entities and offer methods for introspecting them and interacting with them. The other classes of the package are instead completely middleware independent.

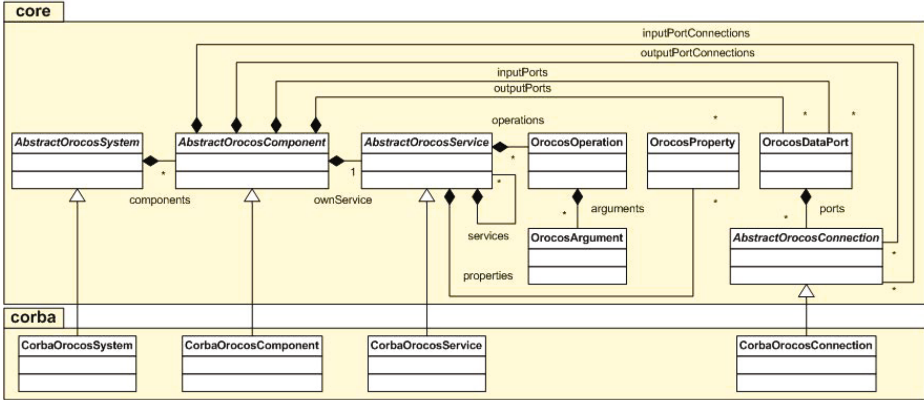


Fig. 3. The JOrocos Architecture

The idea is that the separation of the middleware-independent parts (core package) from the middleware-specific parts (corba package) will allow in future an easier extension of the library in order to provide a support for other middlewares.

The main class of the library is named *AbstractOrocosSystem*. It offers the functionality that allows a developer to connect his software to a running Orocos system, introspect its components and retrieve references to them.

An *AbstractOrocosComponent* is a proxy to an Orocos component and allows the clients to introspect its data ports and its own service. The class offers the operations for creating connections to Orocos ports and writing and reading data on these ports.

The data port of an Orocos component are represented by means of the class *OrocosDataPort*. The interaction with these ports is made available by the class *AbstractOrocosConnection*, which provides the channel that allows the operations of writing data on the output ports and reading data from the input ports.

An *AbstractOrocosService* is a proxy to an Orocos service and offers the functionality for introspecting and invoking its operations and introspecting, reading and writing its properties.

The operations of an Orocos service are represented by means of the class *OrocosOperation*. The properties are instead described by means of the class *OrocosProperty*.

4.2 The SCA-OROCOS Component

In this subsection we will explain how the interaction between Java and Orocos works and how the component *SCA youBot Driver* is implemented. The code reported in the listing 1.1 shows the interfaces of the services provided by the component. The annotation *@Callback* defines the interface that will be used for notifying the events to the *Locomotor*. The annotations *@OneWay* instead means that the invocation of method will be asynchronous.

```
1 public interface retrievingStatus{
2   public double[] getJointsPositions();
3 }
4 @Callback(Notifying.class)
5 public interface SendingCommands {
6   @OneWay
7   public void setJointsPositions(double[] values);
8 }
```

Listing 1.1. The interfaces of the *SCA youBot Driver* services

The listing 1.2 reports the variables declared in the implementation of the *SCA youBot Driver* component.

```
1 @Service(interfaces={retrievingStatus.class,SendingCommands.class})
2 public class SCAYouBotDriver implements retrievingStatus,SendingCommands,Observer{
3   @Property
4   protected String orocosIP;
5   @Property
6   protected String orocosPort;
7   @Callback
8   protected Notifying locomotor;
9   private AbstractOrocosSystem orocosSystem;
10  private AbstractOrodocComponent youBotDriver;
11  private double[] jointsPosition;
```

Listing 1.2. Part of the implementation of the *SCA youBot Driver* component

The class implements the interface *java.util.Obsever* (Observer design pattern [7]), which defines a method for being notified when a new data is available on an Orocos output port. Furthermore the class implements the two interfaces that describe the services.

The first line is a SCA annotation that defines the interfaces of the services of the component. The lines from 3 to 6 declare two SCA properties used for configuring the IP address and the port number of the Corba name service, lines 7 and 8 instead declare a reference to the SCA callback interfaces.

Read and Write Data on Orocos Data Ports. The JOrocos library allows both the operations of reading and writing on an Orocos data port. In order to be executed these operations require a connection between the java client and the Orocos port. Two types of connections are available: data and buffer. On a data connection the reader has access only to the last written value whereas on a buffer connection a predefined number of values can be stored.

The listing 1.3 reports the constructor of the class *SCA youBot Driver* in which the connections to the port are created.

```
1 public SCAYouBotDriver(){
2    orocosSystem = CorbaOrocosSystem.getInstance(orocosIP,orocosPort);
3    orocosSystem.connect();
4    youBotDriver = orocosSystem.getComponent("youBotDriver", false);
5    youBotDriver.createDataConnectionToInputPort("commands", LockPolicy.LOCK_FREE, this);
6    youBotDriver.subscribeToDataOutputPort("joinstStatus", LockPolicy.LOCK_FREE, this, 500);
7 }
```

Listing 1.3. The implementation of the service *SubscribingTojoinstStatusPort*

– Lines 2-3 retrieve a reference to an Orocos running system and create a connection to it.
– Lines 4 retrieves a reference to the *youBot Driver* component.
– Line 5 creates a data connection to the input port *commands* of the Orocos component *youBot Driver*.
– Line 6 creates a data connection to the output port *status* and starts a thread that periodically check if new data are available on the port. This functionality is implemented in JOrocos (methods *subscribeToDataOutputPort* and *subscribeToBufferOutputPort*). In this case a data connection with a lock free policy is created. The third parameter specifies the Observer object that will be notified when new data will be available on the port (in this case it is the component). Finally the last parameter defines the frequency with which the availability of new data on the port will be checked (it is expressed as period in milliseconds).

Once the component is subscribed to the output port it will be notified as soon as a new data will be available by means of the method *update* (inherited from the Observer interface). The implementation of this method is reported in the listing 1.4. It simply stores the new data on the variable *jointsPosition* and notifies the *Locomotor* that new data are available.

```
1 public void update(Observable arg0, Object arg1) {
2   OrocosPortEvent event = ((OrocosPortEvent)arg1);
3   jointsPosition = ((YouBotStatus)event.getValue()).getJointsPosition;
4   locomotor.notify("refresh");
5 }
```

Listing 1.4. The implementation of the method *update*

From this moment the *Locomotor* can retrieve the new data through the operation provided by the interface *retrievingStatus*. Its implementation is reported in the listing 1.5. It simply returns the position of the joints.

```
1 public double[] getJointsPositions() {
2   returns joinstPosition();
3 }
```

Listing 1.5. The implementation of the service *SubscribingTojoinstStatusPort*

The listing 1.6 reports instead the implementation of the operation defined in the service *sendingCommands*. The purpose of this operation is writing the data received from the *Locomotor* to the Orocos output port. The component first notifies the *Locomotor* that the operation is started, then writes the values on the *commands* port and finally notifies the *Locomotor* that the operation is completed.

```
1 public void setJointsPositions(double[] values) {
2   locomotor.notify("busy");
3   youBotDriver.writeOnPort("commands", values, this);
4   locomotor.notify("idle");
5 }
```

Listing 1.6. The implementation of the service *sendingCommands*

The operations of writing and reading data support both simple and complex data types and respectively receive as parameter and return as result instances of the class *Object*. In this context corba introduced two issues:

1. Corba returns references to the requested objects as instances of the class *Any*. Hence the result of a read operation is an *Any* object, whereas we want to return a more general *Object* instance.
2. The cast from *Any* to the right type is possible only by means of the "*Helper*" classes that are automatically generated from the IDL-to-Java compiler. However we cannot know every possible data type a priori and consequently implement all the possible cast in the code of our library.

We solved these two problems by means of the Java reflection. Indeed, in the code of the write and read operations we retrieve the class name from the object that has to be written (in the case of the write operations) or from the *Any* object (in the case of the read operations). Then we use the name of the class for loading at runtime the right "*Helper*" class and using its static method for casting *Any* to *Object* or vice versa. The listing 1.7 shows how our library casts an *Any* to an *Object*.

```
1 // value is the object that has to be written
2 String className = value.getClass().getName(); + "Helper";
3 Class<?> helper = Class.forName(className);
4 Method castMethod = helper.getMethod("insert", Any.class, value.getClass());
5 // the insert method inserts value in the any object received as second parameter
6 castMethod.invoke(null, any,value);
```

Listing 1.7. The *Any* to *Object* cast

5 The Case Study

In order to test the functionality provided by JOrocos we have implemented a simple case study application. It is similar to the scenario introduced in the section 4 but the *Locomotor* component is replaced by a graphical interface (*youBot Monitor* component), which is in charge of plotting the current state of the joints and allowing the user to set the period (inverse of frequency with which the operations of the component is executed) of the *youBot Driver* component.

The *youBot Driver* is currently a dummy component. Indeed we are working on its implementation in the context of the European project BRICS (Best of Robotics [1]), but unluckily it is not ready yet. The dummy component publishes on the output port a set of random values that describe for each joint

position, velocity, current, temperature and error flag (10 bits that provide information about a set of possible errors). For the purpose of the test it doesn't matter whether the values published on the port a real or random. In fact we are only interested in testing the communication between SCA and the Orocos components.

The *youBot Driver* component has a new input port called *period*. When a new data is written on this port the component set its period according to the value of this new data. Due to this new port also the *SCA youBot Driver* has a new provided interface named *settingPeriod*. It provides a service for receiving a new period value from the *youBot Monitor* and writing it on the input port of the *youBot Driver*.

The *youBot Monitor* component has two required interfaces that correspond to the provided interfaces of the *SCA youBot Driver*. It also provides the *notifying* interface, which is used by the *SCA youBot Driver* for notifying its events.

The SCA components and the Orocos component run on two different machines: the last one on board of the robot whereas the other two on the supervisor workstation.

The implementation of the *SCA youBot Driver* component is very similar to the one reported in section 4. The *youBot Monitor* component is instead implemented by using the Java SWING and provides several tabs. In the main tab a set of global information about the state of the joints is showed. This tab also allows the configuration of the *youBot Driver* period. The other tabs instead provide information about a specific joint and plot on a set of charts the trend of the joint values. The graphical interface is depicted in figure 4.

This case study demonstrated how JOrocos makes possible and simple the communication and the cooperation between SCA and Orocos. By writing few lines of Java code we were able to retrieve data from the Orocos output port and set the period of the *youBot Driver* component. Furthermore we didn't have to care about the location of the components on the network (except for setting the IP address and the port of the name service) and the different programming language used for implementing the Orocos component.

(a) Global status (b) Joint 1 charts

Fig. 4. The graphical interface

6 Conclusions

In this paper we have discussed the problem of making possible the cooperation between Service Oriented Architectures (SOA) and Data Flow Oriented Architectures in the robotics field. In particular we have focused our attention on SCA and Orocos, the first a component based SOA and the second an hard real-time component based robotics framework. We have presented a set of architectural mismatches between the two component models and a java-based library, named JOrocos, which allows the developers to bridge these differences by defining proxies components. We have also provided a guideline for the development of these proxies and we have applied it in a case study.

The first mismatch regarded the syntax and the semantics of the messages exchanged between the components in the two frameworks. Here JOrocos provides to the developers the mechanisms for allowing the communication between SCA and Orocos components and translating SCA messages to Orocos messages and vice-versa. However JOrocos doesn't provide the possibility of directly connecting a SCA Service (or Reference) to an Orocos Port. The developer has to define, according to our guideline, a proxy component which provides input to the Orocos component when one of its services is invoked and invoke a service of its client (the SCA component) when the Orocos component produces data on the output port. In this direction a possible improvement will consist of (a) using JOrocos for extending the SCA runtime in order to define a new binding for Orocos and (b) extending the SCA composite designer for supporting this new binding. These extensions will replace the role of the proxy components and will allow the developer to directly connect SCA and Orocos components.

The second mismatch was about the synchronization of the component operations after the action of sending a message. Here JOrocos doesn't provide the possibility of choosing a specific synchronization mechanism. In order to permit both synchronous and asynchronous way, a specific synchronization mechanism has to be implemented in the proxy components. For example, in our scenario we have demonstrated how it is possible to send messages synchronously and asynchronously. Indeed the operation of retrieving the robot status is executed by the *SCA youBot Driver* in a synchronous way, whereas the operation of sending commands in an asynchronous way.

Finally the last mismatch concerns the absence of an hierarchical composition mechanism in Orocos. Here JOrocos allows the developers to leverage on the SCA composition mechanism for encapsulating several Orocos proxies in SCA composites and reusing them in complex and hierarchical systems.

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

The authors would like to thank all the partners of the BRICS project for their valuable comments.

References

1. BRICS - Best of Robotics, http://www.best-of-robotics.org/
2. Brugali, D., Gherardi, L., Riccobene, E., Scandurra, P.: A formal framework for coordinated simulation of heterogeneous service-oriented applications. In: 8th International Symposium on Formal Aspects of Component Software, FACS (2011)
3. Brugali, D., Salvaneschi, P.: Stable aspects in robot software development. International Journal on Advanced Robotic Systems 3(1), 17–22 (2006)
4. Brugali, D., Scandurra, P.: Component-based robotic engineering, part I: Reusable building block. IEEE Robotics & Automation Magazine 16, 84–96 (2009)
5. van de Molengraft, R., Beetz, M., Fukuda, T.: A special issue toward a www for robots. IEEE Robotics Automation Magazine 18(2), 20 (2011)
6. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.: The many faces of publish/-subscribe. ACM Computing Surveys (CSUR) 35(2), 114–131 (2003)
7. Gamma, E.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional (1995)
8. Marks, E., Bell, M.: Service-Oriented Architecture (SOA): A planning and implementation guide for business and technology. John Wiley & Sons (2006)
9. KUKA youBot store, http://youbot-store.com/
10. Mozos, O., Marton, Z.-C., Beetz, M.: Furniture models learned from the www. IEEE Robotics Automation Magazine 18(2), 22–32 (2011)
11. Open Robot Control Software, http://www.orocos.org
12. Service Component Architecture (SCA), http://www.osoa.org
13. SCA Specifications - SCA Assembly Model, http://www.osoa.org/display/Main/The+Assembly+Model
14. Scandurra, P., Riccobene, E.: A modeling and executable language for designing and prototyping service-oriented applications. In: EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011 (2011)
15. Tenorth, M., Klank, U., Pangercic, D., Beetz, M.: Web-enabled robots. IEEE Robotics Automation Magazine 18(2), 58–68 (2011)
16. Yoshimi, T., Matsuhira, N., Suzuki, K., Yamamoto, D., Ozaki, F., Hirokawa, J., Ogawa, H.: Development of a concept model of a robotic information home appliance, aprialpha. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 1, pp. 205–211 (2004)

Safe Autonomous Transport Vehicles in Heterogeneous Outdoor Environments*

Tobe Toben[1], Sönke Eilers[1], Christian Kuka[1],
Sören Schweigert[1], Hannes Winkelmann[1], and Stefan Ruehrup[1,2]

[1] OFFIS e.V., 26121 Oldenburg, Germany
lastname@offis.de
[2] FTW, Vienna, Austria
lastname@ftw.at

Abstract. Autonomous transport vehicles (AGVs) steadily gain importance in logistics and factory automation. Currently, the systems are mainly operating in indoor scenarios at limited speeds, but with the evolution of navigation capabilities and obstacle avoidance techniques, AGVs have reached a degree of autonomy that, from a technical perspective, allows their operation beyond closed work environments. The major hurdle to overcome is to be able to guarantee the required safety level for industrial applications. In this paper, we propose a general architecture for AGVs that formalizes the current safety concept and extends it to vehicles driving at higher speeds in outdoor environments. Technically, the additional safety level is achieved by integrating information from stationary sensors in order to increase the perception of the vehicles.

1 Introduction

Autonomous guided vehicles (AGV) steadily gain importance in logistics and factory automation. Traditionally, those vehicles are working in closed environments on pre-defined driving paths. The driving speed is limited in order to be sure to stop the vehicle as soon as any obstacles is recognized by on-board sensors. A comprehensive overview of the different systems and applications can be found in [25]. With the evolution of navigation capabilities and obstacle avoidance techniques, AGVs have reached a degree of autonomy that allows their operation beyond closed work environments. From a technical perspective, autonomous vehicles can already be used in outdoor environments and working areas that can be entered by people (i.e. not necessarily trained personnel) and other manned vehicles. However, from a safety point of view, these environments pose new challenges, such as the avoidance of moving obstacles, whose appearance cannot be determined in advance. Moreover, changing environment conditions in outdoor scenarios state additional challenges to a safe sensing of the environment. For various scenarios, there are specific technical solutions for a safe operation of automated vehicles. Most solutions address the uncertainty and complexity of the environment by restricting speed and allowing limited

* This work has been supported by the German Federal Ministry of Economics and Technology (BMWi) under the grant 01MA09037.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 61–75, 2012.
© Springer-Verlag Berlin Heidelberg 2012

sensing capabilities of the AGV, which lowers its efficiency on long routes found in outdoor areas. Thus the general safety issues of approaches beyond these settings still remain open. Therefore, we will address a more general approach to system safety by defining an extended AGV architecture with an appropriate and feasible safety concept.

1.1 Safety Aspects for AGVs

Traditionally, automated vehicles are considered as machines and considerations on functional safety follow mechanical engineering principles, where the focus is on component failures and the primary goal is to increase component reliability. Safety risks of the system are assessed after integrating the components into the overall system. If necessary, a stage of risk reduction is performed, e.g. by installation of additional safety devices (e.g. sensors that trigger an emergency shutdown in order to protect human operators). Those devices are usually certified or evaluated, such that their failure probability can be estimated.

According to EN 1525 [12], a dedicated safety standard for automated guided vehicles, safety devices for automated vehicles have to comply to certain safety categories[1] and be able to detect test bodies of a certain size in front of the vehicle. These test bodies are meant to represent an arm or leg of a person standing or lying on the ground. It is the responsibility of the manufacturer to show that the safety devices can fully oversee the route ahead and trigger the breaks on time if necessary. Hence, the range of safety sensors (esp. for obstacle detection) and the stopping distance of the vehicle determine the maximum speed. State-of-the-art industrial laser range finders have a general range of 50m, but a safety range of less than 10m (also called protective zone). Therefore, the maximum speed of AGVs equipped with such sensors is usually in the range of walking speed, which we will call “normal speed” in the following.

At higher speeds an automated vehicle underlies essentially the same safety requirements: It has to avoid collisions with static and moving obstacles. Higher speeds imply a longer stopping distance and thus require a greater sensing range for obstacle detection. Especially for moving obstacles, it is difficult to foresee if a moving object could possibly intersect the planned trajectory. Since the range of on-board safety sensors is limited and moving obstacles might be obstructed, we prepose a safety architecture that uses stationary safety sensors to monitor the areas of operation. This allows the detection of obstacles beyond the range of vehicle’s on-board sensors. Furthermore, we avoid the need for specialized on-board sensors that are not certified for industrial applications.

The additional sensing capabilities by external sensors can be used only in a close coordination with the vehicle control. In this paper, we devise a general safety concept that allows for a formally sound argumentation for a collision-free movement of the automated vehicle. To this end, we decompose the general safety requirement of collision-freedom into a number of sub-requirements that have to be individually fulfilled by the components of the vehicles.

[1] EN 1525:1997 refers to *categories* defined in EN 954-1:1996 [10], which are replaced by *performance levels* in the succeeding standard EN ISO 13849 [14].

1.2 Related Work

Currently, research on autonomous vehicles mainly focusses on how to efficiently navigate in structured and unstructured environments. The most promising results are demonstrated in contests like the DARPA Grand/Urban Challenges (see e.g. [5]). The goal was to build autonomous road vehicles that can cope with an unknown environment and interact with other traffic, but not compliance to industrial safety standards. Therefore, an important property of such vehicles is the sophisticated sensor equipment that allows a precise measurement of the environment (see e.g. [17]). Consequently, the software architecture is designed to support fast sensor data processing, interpretation, object tracking and online mapping of the environment. Safety aspects are already considered for the involved algorithms, e.g. for context recognition [28,11,23] or motion planning [22,13,1,21], but there is typically no dedicated safety concept that can be used to validate or even guarantee a certain safety level of the overall system.

Our aim is to develop a general safety concept for autonomous movements. A number of basic safety notions for driverless vehicles are given in [4], from which we adopt the requirement that the vehicle may not harm by action (though we cannot guarantee to be harmless by inaction, cf. Sect. 3). In the SAMS project [16], a dedicated on-board safety component in the context of autonomous robotics is modeled and verified by means of formal methods [27]. This component focuses on the calculation of dynamic safety zones, which are protected by a laser scanner, depending on the vehicle's intended maneuvers.

In our setting, we use additional external sensors to increase the awareness of the vehicles, that is, we combine mobile and stationary sensors. Our proposed architecture is intended to extend a common AGV design, where safety functions are implemented separately from navigation and control components. In this respect our concept differs from system architectures proposed for the aforementioned autonomous road vehicles [17] and from architectures for cooperative vehicle-to-vehicle applications (e.g. platooning) [7]. The latter architecture is divided into layers according to the the communication needs with the cooperating counterpart, whereas our architecture consists of operational and safety layers.

In the automotive domain, the usage of road side infrastructures can also be seen as a combination of mobile and stationary sensors. Safety aspects in this particular context are for example addressed in the SafeSpot [19] project, where an intersection application was developed that tracks pedestrians and cyclists at an intersection by laser scanners and and warns the driver against critical right turns. While use of stationary sensors is similar to our approach, the system issues warnings to a driver. In this setting, failures of the external systems and loss of messages are tolerable, since the driver still has the responsibility to act accordingly. In our case, we have to make sure that the autonomous vehicle operates safely in the case of external failures.

1.3 Structure

The paper is organized as follows: Section 2 describes the current state of the art in the field of autonomous transport vehicles. Section 3 describes the safety

concept for efficient AGVs and their requirements, followed by the system architecture and the safety analysis of the architecture. Based on the architecture, Section 4 presents the technical concept, including a brief overview of a sensor layout as well as extensions towards a more efficient trajectory planning based on obstacle classification and hazard prediction. Section 5 concludes the paper.

2 State of the Art

In Fig. 1 we illustrate the basic architecture of a classical AGV. This architecture is intended to be a high-level description which covers a large class of existing implementations. The functional behaviour of the vehicle is realized by the components within the *operational layer* at the bottom. After a transport task has been assigned by the control centre, the Navigation component calculates a suitable driving path. In AGVs with a higher degree of autonomy, the Trajectory planner is responsible to compute a feasible driving trajectory. Classic guidance technology along a fixed track (e.g. via transponders) does not require a trajectory planner and reduces the navigation to a simple selection between alternative routes, if applicable. Finally, the vehicle follows the desired track or trajectory by using the Actuator control component which actually controls the hardware actuators of the vehicle. Note that these components need to know whether the vehicle is on the designated position or path, which is provided by a dedicated component for (absolute or relative) Localization.

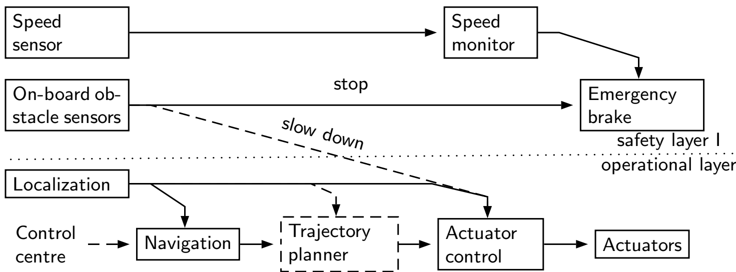


Fig. 1. Control flow between components of a standard architecture for AGVs

A dedicated *safety layer* is responsible for collision avoidance. To this end, Onboard obstacle sensors are used for detecting obstacles within the driving path. Technically, this is typically carried out by tactile bumpers or LIDAR based sensors (cf. Sect. 4.1). The presence of an obstacle within a pre-defined *protective zone* in front of the vehicle triggers the Emergency brake. In order to reduce the number of brakings, some systems also implement a more preventive strategy which slows down the vehicle if there is an obstacle within the *warning zone* (which strictly covers the protective zone). However, such comfort mechanisms are typically not part of the safety concept itself. The dimensions of these zones

are defined such that the vehicle is able to stop before hitting the obstacle. As this property relies on a predefined maximal speed of the vehicle, a combination of Speed sensor and Speed monitor constantly checks whether the vehicle is operating below this maximum. If not, the vehicle is stopped. Obviously, the maximum speed depends on the employed technology for obstacle detection. In current systems, the speed is typically limited to walking speed (i.e. less than 3m/s). In the following, we call this the "normal" speed.

3 Safety Concept for Efficient AGVs

The goal of building a safe and efficient AGV for outdoor environments (as sketched in the introduction) corresponds to the following three (abstract) top-level requirements, namely

1. The AGV must be able to perform transportation tasks.
2. The AGV must be able to drive faster than normal.
3. The AGV must not collide when moving.

The first (functional) requirement corresponds to the general ability of executing the desired transport tasks, which depend on the actual application and typically include e.g. the ability to accept new tasks, to autonomously drive to the desired location and to pick up the freight. The second (non-functional) requirement states that the classical driving speed of indoor AGVs is not sufficient for outdoor applications, but rather a higher speed (called "fast" in the sequel) is required for an efficient application. The third (safety) requirement forbids collisions with any kind of obstacles. Collisions are only accepted if the vehicle is not moving, as such collisions cannot be avoided by the vehicle itself. There is a clear link between the latter two requirements, as in particular driving at a higher speed must not leads to collisions.

In Subsection 3.1, we describe our overall safety concept that ensures collision-freedom. We then present in Subsection 3.2 a system architecture that allows for an implementation of the (non-)functional requirements and in particular supports the safety concept. Finally, Subsection 3.3 identifies further safety requirements that arise due to the technical realization of the safety concept.

3.1 Safety Concept

Collision-avoidance for classical AGVs is ensured by three aspects, namely

1. the vehicles' speed never exceeds normal speed,
2. the absense of obstacles ensures collision-freedom, and
3. the presence of obstacles forces the vehicle to stop.

More abstractly, the vehicle is always in one of the two modes *driving* or *standing*, and the obstacle sensor is responsible to trigger a timely transition from the first to the latter mode. Our safety concept for safe and efficient AGVs is a natural generalization of this principle. That is, we limit the AGV's behaviour with

respect to a dedicated set of operational *safety modes*, and these modes are set at run-time according to the current environmental situation and then directly constrain the physical execution of movements. One particular advantage is that errors in intermediate decisions components are masked and thus can not lead to collisions. By this, certain components of the system become irrelevant for the overall safety analysis. We will discuss which of the system components remain safety-relevant in Sect. 3.3.

For our application, we define the three safety modes stop, normal and fast. Intuitively, a vehicle that is not moving corresponds to the fact that mode stop is active. The other two modes denote a limitation to normal and fast driving speed, respectively. We distinguish between the mode that is currently *set*, and the mode that is currently *active*. Note that these two modes are not necessarily the same, basically as (de-)acceleration takes some time.

We call the vehicle *safe* as long as it has a certain positive distance to all other obstacles. As a safe vehicle is in particular collision-free, requirement (3) is implied by *"The vehicle is always safe or in mode stop"*. This requirement in turn can be ensured by the combination of the following constraints:

3.1 Initially, mode stop is set and active.
3.2 Always one of the modes stop, normal or fast is active.
3.3 The modes normal and fast become active only if they are set.
3.4 Setting a mode leads to its timely activation, namely
  (a) If normal is active and stop is set, stop is active after t_n time units.
  (b) If fast is active and normal is set, normal is active after t_f time units.
  (c) If fast is active and stop is set, stop is active after $t_f + t_n$ time units.
3.5 Setting a mode guarantees safety for a pre-defined time, namely
  (a) If normal is set, driving with normal speed is safe for t_n time units.
  (b) If fast is set, driving with fast speed is safe for $t_f + t_n$ time units.

The derived requirements can be classified into three categories, namely *speed control* (3.1, 3.2, 3.3), *speed reduction* (3.4), and *speed clearance* (3.5). Note that t_n is the time to stop the vehicle when driving at normal speed and t_f is the time to decelerate the vehicle from fast speed to normal speed.

To prove that the satisfaction of requirements 3.1 to 3.5 are actually strong enough to ensure the top-level safety requirement 3, we formalized the requirements in the linear temporal logic (LTL [18]) as stated in the following and analyzed the implication by means of model-checking techniques[2].

In LTL, system states are expressed by the valuation of a set of predefined atomic propositions. Besides the classics boolean operations, the evolution of these propositions over time can be formalized by temporal operators. We use a discrete time version of LTL where time is modeled a sequence of equidistant steps. For example, the expression '$\mathbf{X}\,\phi$' states that the proposition ϕ holds in the next system step, while for '$\mathbf{G}\,\phi$' the proposition has to hold forever from now on. Moreover, '$\mathbf{D}_t\,\phi$' requires ϕ to hold during the next t steps, while '$\mathbf{F}_t\,\phi$' requires ϕ to finally hold at least after t steps.

[2] The implementation can be found at `http://vhome.offis.de/tobe/sars2011/`.

We use propositions $\mathsf{act}(m)$ and $\mathsf{set}(m)$ for each mode $m \in \{\mathsf{stop}, \mathsf{normal}, \mathsf{fast}\}$ to state that mode m is currently active or set, respectively. The proposition safe symbolically denotes a safe state of the vehicle (see above). Then the top-level requirement (3) and the decomposed subgoals can be formalized to corresponding LTL formulae ϕ_3 and $\phi_{3.x}$ as follows:

$$\phi_3 \;=\; \mathbf{G}\;\big(\neg\mathsf{safe} \rightarrow \mathsf{act}(\mathsf{stop})\big)$$

$$\begin{aligned}
\phi_{3.1} &= \;\;(\mathsf{set}(\mathsf{stop}) \wedge \mathsf{act}(\mathsf{stop}))\\
\phi_{3.2} &= \mathbf{G}\;\big(\mathsf{act}(\mathsf{stop}) \vee \mathsf{act}(\mathsf{normal}) \vee \mathsf{act}(\mathsf{fast})\big)\\
\phi_{3.3} &= \mathbf{G}\;\big((\neg\mathsf{act}(\mathsf{normal}) \wedge \mathbf{X}\,\mathsf{act}(\mathsf{normal}) \rightarrow \mathsf{set}(\mathsf{normal}))\;\wedge\\
&\qquad (\neg\mathsf{act}(\mathsf{fast}) \wedge \mathbf{X}\,\mathsf{act}(\mathsf{fast}) \rightarrow \mathsf{set}(\mathsf{fast}))\big)\\
\phi_{3.4} &= \mathbf{G}\;\big((\mathsf{act}(\mathsf{normal}) \wedge \mathsf{set}(\mathsf{stop}) \rightarrow \mathbf{F}_{t_n}\,\mathsf{act}(\mathsf{stop}))\;\wedge\\
&\qquad (\mathsf{act}(\mathsf{fast}) \wedge \mathsf{set}(\mathsf{normal}) \rightarrow \mathbf{F}_{t_f}\,\mathsf{act}(\mathsf{normal}))\;\wedge\\
&\qquad (\mathsf{act}(\mathsf{fast}) \wedge \mathsf{set}(\mathsf{stop}) \rightarrow \mathbf{F}_{t_n+t_f}\,\mathsf{act}(\mathsf{stop}))\big)\\
\phi_{3.5} &= \mathbf{G}\;\big((\mathsf{set}(\mathsf{normal}) \rightarrow \mathbf{D}_{t_n}(\mathsf{act}(\leq\mathsf{normal}) \rightarrow \mathsf{safe}))\;\wedge\\
&\qquad (\mathsf{set}(\mathsf{fast}) \rightarrow \mathbf{D}_{t_n+t_f}(\mathsf{act}(\leq\mathsf{fast}) \rightarrow \mathsf{safe}))\big)
\end{aligned}$$

$$\text{where}\quad \begin{aligned}\mathsf{act}(\leq\mathsf{normal}) &:= \mathsf{act}(\mathsf{stop}) \vee \mathsf{act}(\mathsf{normal})\\ \mathsf{act}(\leq\mathsf{fast}) &:= \mathsf{act}(\leq\mathsf{normal}) \vee \mathsf{act}(\mathsf{fast})\end{aligned}$$

Note that the '**X**' expressions of the active modes in clause $\phi_{3.3}$ do not express that the mode switching happens in one time units. The expression rather captures the moment in time where the new mode becomes active and requires that the corresponding mode must be *set* at that point in time. We applied the model-checking engine NuSMV [8] to show that

$$(\phi_{3.1} \wedge \phi_{3.2} \wedge \phi_{3.3} \wedge \phi_{3.4} \wedge \phi_{3.5}) \rightarrow \phi_3$$

is a tautology, that is, whenever all sub-goals are satisfied by a system, the satisfaction of the top-level requirement is guaranteed.

3.2 System Architecture

Figure 2 shows a system architecture which conservatively extends the basic architecture as discussed in Sect. 2. In the following, we will lay out how this design supports our safety concept. To this end, we will map the safety requirements 3.1 to 3.5 from the previous section to the architecture.

We give an overview of the architecture first. The *operational layer* and the *safety layer I* basically correspond to the classical architecture from Fig. 1. In particular, it comprises components for Navigation and Trajectory planning in order to support a higher degree of autonomy than classical AGVs. A further *safety layer II* is responsible to ensure safe driving at a faster speed. To this end,

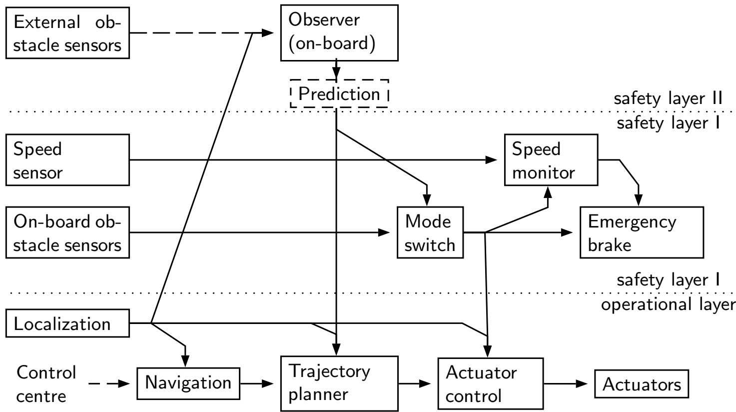


Fig. 2. Control flow between components of the extended architecture

External obstacle sensors (cf. Sect. 4.1) are used to monitor the driving path in front of the vehicle. This information is communicated to an on-board Observer component, which, taking the current position of the vehicle into account, then decides whether driving at fast speed is safely possible. The information from both On-board sensors and Observer is given to the Mode switch component, which sets the suitable safety mode for Speed monitor, Emergency brake, and Actuator control. In turn, these components adjust their speed limit to the current mode. In particular, the emergency brake is triggered when mode stop is set.

We now explain which components are responsible to ensure the derived safety requirements from the previous section. The requirements for *speed control*, i.e. 3.1, 3.2, and 3.3, are realized by the Mode switch, Actuator control and Speed monitor components. In particular, the output of the Mode switch must be consistent with the input from the sensors, and the Speed monitor must trigger the Emergency brake whenever the vehicles' speed erroneously exceeds the active mode. The requirement for *speed reduction* (3.4) addresses the timely activation of mode reductions. Switching from fast to normal in time t_f is a property of the Actuator control and the Actuators themselves. Switching from normal to stop in time t_n basically corresponds to the braking distance of the vehicle. Physical conditions of the outdoor environment affecting the braking distance apply here, and realistic values has to be assessed depending on worst-case assumptions of the actual working environment.

The requirements for *speed clearance* (3.5) is the most involved constraints. Setting mode normal corresponds to the classical safety mechanism based on the on-board sensors (i.e. driving with normal speed is safely possible when no obstacles are in the protective zone). Setting mode fast is only valid if driving with fast speed is safely possible for the next $t_f + t_n$ time units. This property has to be ensured by the Observer component (cf. Subsection 3.3 below), based on the vehicles current position and the environment data of external sensors.

Intuitively, using external sensors within safety layer II corresponds to an extension of the classical protective and warning zones, that is, the additional

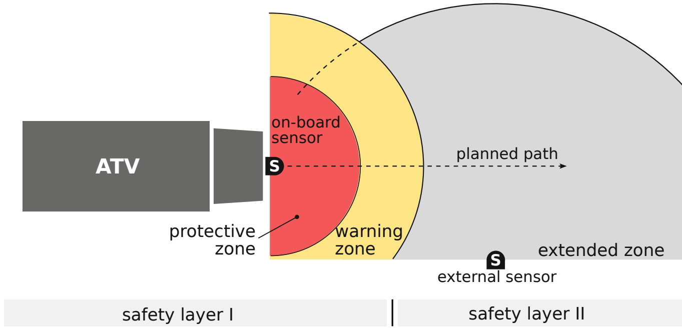


Fig. 3. Zones covered by safety sensors (S) and their respective safety layer in the system architecture (cf. Fig. 2)

sensors increase the recognition of obstacles in the dimension of space and time (cf. Fig. 3). Setting mode fast then corresponds to the fact that no obstacle is in the *extended zone*. This however can be a very conservative implementation of requirement 3.5(b) as not every obstacle in this zone necessarily poses a safety risk. To this end, an (optional) Prediction component can be used to estimate the movements of the obstacle for the near future and then to decide whether a mode reduction is required. We will discuss prediction in more detail in Sect. 4.4.

3.3 Safety Analysis

Given the architecture above, we identify a set of system components as safety-relevant, that is, their malfunction may lead to a violation of the top-level safety requirement. For the driving at normal speed, these are exactly those components which are located within safety layer I. We identify two kinds of information flow in this layer, namely on the one hand the *mode setting* starting from the On-board sensors via the Mode switch to the Emergency brake, and on the other hand the *speed monitoring*, starting from the Speed sensor via the Speed monitor to the Emergency brake. For the safety assessment within this paper, we focus on the requirements regarding the *speed clearance*, because the constraints on *speed control and -reduction* can be ensured and verified with standard techniques [26]. The same applies to the correctness of the information flow itself.

Speed clearance for mode normal is formalized by requirement 3.5(a) and has to be guaranteed by the on-board obstacle sensors. We will elaborate on reliable obstacle detection using classical on-board sensors in Sect. 4.1 below. Speed clearance for mode fast corresponds to requirement 3.5(b), and we introduced the component Observer to realize this requirement. It comes along with an additional information flow from Localization and External obstacle sensors to the Observer and further via Mode switch to the Emergency brake or Actuator control. Again, we will discuss the technical realization in Sect. 4.1 below. Besides a reliable detection of obstacles, we are confronted with two additional issues:

1. The information from the external stationary sensors has to be transferred via (in general unreliable) communication channels. Thus the Observer cannot rely on information being always available; it has to deal with "missing" information in a way that it has to assume non-existent external obstacle sensors for the time the communication is not possible. In addition, the integrity of the communicated information has to be ensured by using suitable protocols, typically involving check-sums and timestamps.
2. A correct localization of the vehicle becomes safety-relevant. This is as the clearance from the external sensors is always bound to the current position of the vehicle. Hence, a reliable localization method has to be employed. One approach is to use the detection of the vehicle from the perspective of the external sensors in order to validate the internally estimated position of the vehicle. Another possibility is the redundant layout of on-board localization.

4 Technical Concept

The technical concept shows how the proposed architecture can be supported and implemented. The basis are reliable sensors for the external monitoring, as described in Subsection 4.1. Their data is communicated to the on-board Observer, which decides if a fast speed is safe. Its decision making is described in Subsection 4.2 based on a conservative estimation of obstacle movements. A better estimation can be achieved by classification of obstacles, as described in Subsection 4.3. Based on this classification, Subsection 4.4 sketches, how the movement of objects can be predicted for a more efficient trajectory planning.

4.1 Safety Sensors

Most AGVs use tactile or optical safety sensors that support safety layer I (cf. Fig. 2). Tactile sensors ("bumpers") respond to a physical deformation when touching an obstacle. The range of these sensors is usually limited. Optical sensors such as laser scanners can detect obstacles at a larger distance and without touching them. Typical devices allow to configure different ranges or zones, such that the vehicle is slowed down when an obstacle is detected within a larger distance and stopped using the emergency break only in case of a close obstacle.

Optical sensors can be used as well to support safety layer II (cf. Fig. 2). When used as external and static sensors, a reference body can be set up in almost any distance and thus exploit the effective range of the sensors. There are the following advantages of static sensors over mobile on-board sensors: Some sensors are completely excluded from mobile use because of their principle of measurement (e.g. motion detectors). Other sensors are prone to direct sunlight or other influences in an outdoor environment. Moreover, some sensors, e.g. PMDs (photonic mixing devices) or ultrasonic sensors, may interfere with a sensor of the same type if the emitted signal of another device is interpreted incorrectly. In a static layout, such sensors can be placed in a way that minimizes negative environmental influence and interference.

Therefore it should be part of the roll-out of the complete system to take these influencing factors into account, while providing coverage of areas that need consistent observation for safety or efficiency reasons. An (external) static sensor setup serves already as an implementation of requirement 3.5b. Due to the arguments above, we see a high potential for existing safety sensors to be employed for such external monitoring tasks, even in outdoor settings.

Also further data processing steps, as described in the next subsections, can benefit from a static sensor setup. Some simple but effective algorithms such as background subtraction can be used to detect obstacles or moving objects if a static background is given. A further advantage is the known position and orientation of each stationary sensor.

4.2 The Observer Component

The Observer component evaluates the data that is communicated from external sensors to the AGV. It is integral part of the extended architecture and responsible for ensuring that the fast mode is only active if the AGV is in a safe state. In the following we sketch its operation principle in a conservative mode, which we think can be validated with state-of-the-art methods. Further improvements require object classification and prediction as specified in subsequent subsections.

We assume that external sensor infrastructure communicates obstacle information to the vehicle repeatedly with an update interval of Δt. While the reliability of sensor measurements are assumed to meet industrial safety requirements, no assumptions are made on the (wireless) communication with the vehicle.

When receiving the external sensors' data, the Observer has to decide if the fast mode can be activated or remain active by calculating the reachability region of the AGV itself and the reachability region of static and moving obstacles. Reachability regions (or reachable sets, see e.g. [20]) cover all possible trajectories or locations within a certain time interval.

1. The reachability region R_{AGV} of the AGV is calculated for the next $t_n + t_f + \Delta t$ time units (update interval plus deceleration and stopping time). It should be based on the current speed, the current location and an overestimation of the vehicles dynamics, such that failures in lateral control or break actions are covered. R_{AGV} corresponds to the existence regions in [20]. Note that a precise localization of the AGV is vital; a loss of precision has to result in leaving the fast mode.
2. The reachability region R_{obs} of all obstacles is calculated. A 2D mapping of the obstacles, e.g. the contour obtained from a laser scan, is sufficient as input. In case of a known static background (walls, fences), the contour gives the border of the reachable set. Other obstacles have to be treated as potentially moving, i.e. a safety margin has to be added to the reachable set R_{obs}. This safety margin must cover all possible movements for the duration of $t_n + t_f + \Delta t$. A margin must also be added to the boundaries of the sensing range, where moving obstacles can possibly enter the observed region.

   Since the external sensors are static, the following simplification is possible: A sensor covers a fixed predefined region T as well as an additional

safety margin S. If the covered area $T + S$ is free of obstacles, $R_{\text{obs}} = \overline{T}$ is sent to the AGV. Note that Δt has to be fix then, as S depends thereon.
3. The observer checks whether $R_{\text{AGV}} \cap R_{\text{obs}} = \emptyset$. If the intersection of both regions is empty, then the fast mode can be activated or remain active for Δt time units. Otherwise and in case of a missing update it is set to normal.

The calculated reachability regions grow with the time interval Δt, such that non-empty intersections become more likely and trajectories are rendered unsafe. Distinguishing static obstacles from moving obstacles can help to reduce safety margins. For moving obstacles, the concept of stochastically reachable sets [2] was introduced. Both require obstacle classification as discussed in the following.

4.3 Fusion of Sensor Data and Obstacle Classification

With a conservative calculation of reachability sets, an AGV's trajectory can be regarded as unsafe even if there is no risk of collision. Thus, the goal is to identify moving obstacles and their movement direction and distinguish them from static obstacles in order to obtain more precise reachability sets. These sets are usually smaller and allow to plan better trajectories. Obstacle classification should should provide higher levels of information, e.g. information about objects instead of collections of distance measurements. This object data is further processed in the Prediction component, which is described in the next subsection. Obstacle classification can be facilitated by high-end single-sensor solution such as 3D laser range finders. For most use cases of AGVs, however, such solution is too costly and has to be replaced by industrial sensors, or combinations of those. Combinations of sensors , e.g. laser range finders, which are well suited for precise contour finding, with other optical sensors that support object classification, help to overcome limitations of a single sensing technology.

When combining sensor data, the error margins and dependencies have to be preserved. Especially sensor measurements in outdoor scenarios are tackled by environmental influences that can affect sensors of the same class. It is vital so safety-relevant processes to eliminate false negatives. If this cannot be guaranteed, the system has to fall back into the appropriate mode for safe operation.

Existing systems for sensor data fusion target the improvement of the object detection. However most of the works do not consider the evidence of the measurements of the participating sensors and the sensor data processing in the results [3]. Hence, these systems cannot or only rarely give any consideration about the quality of their results concluding in a binary assertion about the detected obstacles. While this is acceptable in non-safety-critical applications, the evidence about the non-presence of an obstacle in safety-critical systems is significant. A number of mechanisms have been proposed for handling this uncertainty [6]. They include Fuzzy Logic, Bayesian Networks, Hidden Markov Models and the Dempster-Shafer theory. As also stated in [6] a hybrid approach to process the uncertainty and perform a context reasoning should be preferred.

To support of a wide variety of sensors we use an architecture consisting of three logical layers. Simple sensors (without interpretation logic) such as laser

range finders send their measurements to the lowest layer, called raw data layer. More intelligent sensors that can already detect and separate contours of an object send their measurements to the second layer, the feature layer. Sensors that can already detect and classify objects send their data to the highest layer, called the object layer. The raw data layer is responsible to extract features from raw data and propagate them with the measurement uncertainty to the feature layer. The feature layer detects and classifies objects formed by the detected features. It also propagates uncertainty about detected objects. In the object layer, the detected objects are tracked and enhanced with their speed and direction. The object layer appends uncertainty of the tracking to further processing. This object classification and tracking enables a prediction of future movements.

4.4 Hazard Prediction

An important aspect in the efficient operation of an AGV is the planning of trajectories. Autonomous vehicles can dynamically (re-)plan their trajectories, which allows to chose an alternative, if the intended trajectory is not safe or requires a stop. The assessment of trajectories w.r.t. potential hazards is the task of the Prediction component. Though the prediction is not required in the safety architecture, it plays a vital role in improving the overall efficiency. Especially outdoor scenarios are often characterized by a mixed traffic environment involving uncooperative users. If the dynamics of such uncooperative users (vehicles or pedestrians) can be anticipated rather than making worst-case assumptions of their movement, a more efficient dynamic trajectory planning is possible. Based on object detection and classification, the dynamics of other users are predicted in order to identify possibly critical situations and resolve conflicts in a safe way.

The prediction of movement can be generally performed in three different ways [15]: Nominal, probabilistic, and worst-case. The nominal movement is an extrapolation of the past movement, while the worst-case movement model returns reachability sets (cf. Subsection 4.2). The probabilistic model adds probabilities to these positions (e.g. [2]). For pedestrians a worst-case prediction is adequate because of their agility. For vehicles a probabilistic model is recommended that includes changes in speed and direction, but only to a certain realistic extent. Such reachability sets of possible movements for a certain time interval can then be checked for conflicts with the AGV's planned trajectory.

In case of a conflict, the movement of the AGV has to be adapted, such that it results in a safe trajectory. Depending on the dimension of intervention (longitudinal or also lateral control), the prediction triggers the mode switch or the trajectory planner (Fig. 2). An adaptive trajectory planning offers higher efficiency gains than a speed adaption, but also involves further challenges in the realization of the prediction component. This also depends on the quality of available information given by external and on-board sensors. If all moving objects in the environment can be reliably classified as either pedestrians or vehicles, a better prediction of their movement is possible. In case of uncertainty, the worst-case prediction has to be applied, which results in lower speed or more conservative (re-)planning of trajectories.

5 Conclusion

The presented architecture shows how the existing safety case for autonomous transport vehicles can be extended to heterogeneous environments. This extension uses external safety sensors and allows to overcome limitations by relying on a vehicle's on-board sensors only. By extending an established architecture, we can use a well-understood safety case as a fallback if the external infrastructure cannot guarantee safety or fails. Furthermore, we avoid to introduce software-intensive components into the basis architecture.

We have formalized the safety requirements and derived sub-goals, for which we showed completeness and also mapped them to the system components of the architecture. As of yet, the requirements focus on the internal switch control. In a further step, a (probabilistic) formalization of the sensory components is to be integrated, e.g. based on the modeling framework in [23,24]. In general, the implementation of this architecture should be embedded in a safety lifecycle and accompanied by a preceding hazard analysis (e.g. using a generic hazard list [9]) to determine areas of operation that have to be monitored by external sensors.

We have shown that the system architecture with a conservative obstacle detection allows for a fulfillment of the main safety requirements. We have sketched the technical aspects to illustrate the feasibility of including external safety sensors into an overall safe system for a conservative setting. The concept still has a potential for providing more efficient trajectories in the presence of moving obstacles while preserving the safety constraints. However, in order to tap this potential, considerable work has to be done to safely classify objects in the environment, plan trajectories and assess their risk automatically and eventually choose a safe and efficient path.

References

1. Alami, R., Krishna, K., Simeon, T.: Provably Safe Motions Strategies for Mobile Robots in Dynamic Domains. In: Autonomous Navigation in Dynamic Environments. STAR, vol. 35, pp. 85–106. Springer, Heidelberg (2007)
2. Althoff, M., Stursberg, O., Buss, M.: Stochastic reachable sets of interacting traffic participants. In: 2008 IEEE Intelligent Vehicles Symposium, pp. 1086–1092 (2008)
3. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. IJAHUC 2(4), 263–277 (2007)
4. Benenson, R., Fraichard, T., Parent, M.: Achievable safety of driverless ground vehicles. In: ICARCV, pp. 515–521. IEEE (2008)
5. Berger, C., Rumpe, B.: Autonomous Driving - Insights from the DARPA Urban Challenge. IT - Information Technology 50(4), 258–264 (2008)
6. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing (June 2009)
7. Caveney, D.: Hierarchical software architectures and vehicular path prediction for cooperative driving applications. In: 11th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC 2008), October 2008, pp. 1201–1206 (2008)

8. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
9. Eilers, S., Peikenkamp, T., Rührup, S., Schweigert, S., Toben, T., Winkelmann, H.: Eine Generische Gefährdungsliste für Fahrerlose Transportfahrzeuge in der Intralogistik. In: Automatisierungs-, Assistenzsysteme und eingebettete Systeme für Transportmittel, pp. 245–259. ITS Nds. e.V, Braunschweig (2011)
10. EN 954-1:1996, Safety of machinery - Safety-related parts of control systems (1996)
11. Ess, A., Leibe, B., Schindler, K., Gool, L.J.V.: Moving obstacle detection in highly dynamic scenes. In: ICRA, pp. 56–63. IEEE (2009)
12. European Committee for Standardization (CEN): Safety of industrial trucks - Driverless trucks and their systems; German version EN 1525 (1997)
13. Fraichard, T., Asama, H.: Inevitable collision states - a step towards safer robots? Advanced Robotics 18(10), 1001–1024 (2004)
14. ISO EN 13849-1:2006 Safety of machinery - Safety-related parts of control systems (2006)
15. Kuchar, J.K., Yang, L.C.: A review of conflict detection and resolution modeling methods. IEEE Trans. on Intelligent Transportation Systems 1, 179–189 (2000)
16. Lüth, C., Frese, U., Täubig, H., Walter, D., Hausmann, D.: SAMS Sicherheitskomponente für Autonome Mobile Serviceroboter. In: VDI-Bericht 2012. VDI (2008)
17. Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., Thrun, S.: Junior: The stanford entry in the urban challenge. J. Field Robot. 25, 569–597 (2008)
18. Pnueli, A.: The Temporal Logic of Programs. In: 18th Symposium on Foundations of Computer Science, pp. 46–57 (October 1977)
19. Safespot: cooperative vehicles and road infrastructure for road safety, http://www.safespot-eu.org
20. Schmidt, C., Oechsle, F., Branz, W.: Research on trajectory planning in emergency situations with multiple objects. In: IEEE Intelligent Transportation Systems Conference (ITSC 2006), pp. 988–992 (2006)
21. Seward, D., Pace, C., Agate, R.: Safe and effective navigation of autonomous robots in hazardous environments. Auton. Robots 22, 223–242 (2007)
22. Thorpe, C., Carlson, J., Duggins, D., Gowdy, J., MacLachlan, R., Mertz, C., Suppe, A., Wang, B.: Safe Robot Driving in Cluttered Environments. In: 11th Int. Symposium on Robotics Research (ISRR 2003), pp. 271–280 (2005)
23. Toben, T.: A Formal Model of Reliable Sensor Perception. In: Lukowicz, P., Kunze, K., Kortuem, G. (eds.) EuroSSC 2010. LNCS, vol. 6446, pp. 94–107. Springer, Heidelberg (2010)
24. Toben, T., Rakow, J.H.: Safety and Precision of Spatial Context Models for Autonomous Systems. In: 1st ETAPS Workshop on Hybrid Autonomous Systems (HAS 2011). ENTCS (2011) (to appear)
25. Ullrich, G.: Fahrerlose Transportsysteme. Vieweg + Teubner (2011)
26. Vincoli, J.: Basic Guide to System Safety, 2nd edn. Wiley, New York (2006)
27. Walter, D.: A Formal Verification Environment for Use in the Certification of Safety-Related C Programs. Ph.D. thesis, Universität Bremen (2010)
28. Wardziński, A.: The Role of Situation Awareness in Assuring Safety of Autonomous Vehicles. In: Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166, pp. 205–218. Springer, Heidelberg (2006)

Adaptive Autonomous Systems – From the System's Architecture to Testing

Franz Wotawa

Technische Universität Graz,
Institute for Software Technology,
Inffeldgasse 16b/2, 8010 Graz, Austria
wotawa@ist.tugraz.at
http://www.ist.tugraz.at/

Abstract. Autonomous systems have to deal with situations where external events or internal faults lead to states, which have not been considered during development. As a consequence such systems have to have knowledge about themselves, which has to be used to reason about adaptions in order to fulfill a given goal. In this paper we present a control architecture that combines a the sense-plan-act paradigm and model-based reasoning. The latter is used to identify internal faults. Beside discussing the control architecture we also briefly explain advantages of model-based diagnosis where there is a shift from providing control programs to developing models. Consequently, testing of models becomes an issue. Therefore, we also introduce basic definitions related to testing models for diagnosis and discuss the involved challenges.

Keywords: Adaptive systems, model-based reasoning, testing model-based systems.

1 Introduction

Truly autonomous systems like mobile robot have to fulfill given tasks even in case of internal faults or external events potentially preventing the robot from reaching its goal. Such an autonomous robot has to detect a misbehavior by its own. Therefore, the robot has to have capabilities to reason about its state and all the consequences of the state. One consequence might be that a sensor signal cannot longer be trusted or that certain reconfigurations of the hardware and software have to be performed. The ability of reasoning about itself is even more required in cases where a robot has to deal with non-foreseen situations during the development of the robot.

Today almost no available robot is able to deal with internal faults or not previously known external events in an intelligent way. The reasons are (1) the increasing complexity of systems dealing with faults in an automatic fashion, (2) the complexity of the environment, i.e., the real physical world, of an autonomous and mobile robot, and (3) difficulties to enumerate all potential faults and interaction scenarios in advance, e.g., during the development phase of the

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 76–90, 2012.
© Springer-Verlag Berlin Heidelberg 2012

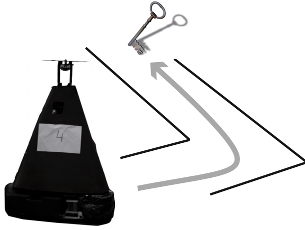


Fig. 1. Fetch the key scenario

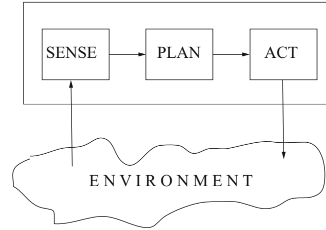


Fig. 2. Sense-plan-act control architecture

robot. The latter prevents using simple solutions like rule-based systems. Moreover, there is also the problem of testing intelligent systems especially when using neural networks and other potentially non-deterministic approaches.

In this paper we present a software architecture that allows for constructing truly autonomous systems. We use model-based reasoning techniques for enabling a robot reasoning about itself. We state the theory behind the approach. But the paper focuses more on the architecture and its components, and on testing. In the following we illustrate the underlying idea using an example scenario where a robot has to fetch a key lying on a floor (see Fig. 1). In many cases the used underlying control architecture of robots follows more or less a traditional sense-plan-act architecture depicted in Fig. 2. In this architecture the robot computes a plan using the given sensor inputs, its internal state, and the given goal. A plan in this terminology is nothing else than a sequence of actions that are executed sequentially. The execution of the actions potentially changes the state of the environment, which is sensed again. In case of failing actions or situations where a plan cannot longer be followed re-planning is performed.

A plan for our key fetching task would include actions for moving forward, turning left, and moving forward again until the robot reaches the key, i.e., $\langle forward, left, forward \rangle$. In a perfect world such a plan can be correctly executed. Let us assume now that in the first $forward$ action there is a problem with the robot's drive. The attached sensor of a wheel is returning that the wheel is not rotating. Hence, the action cannot be successfully continued and the robot might not be able to fulfill its objective anymore because of lack of functioning actuators allowing a robot to move from one position to another. However, the reason for the sensor signal might not be that the motor is not working anymore. It is also possible that the sensor is faulty but distinguishing this two explanations requires self-reasoning capabilities and a certain degree of redundancy. In this example, a robot that is able to identify wrong sensor information would be able to reach the goal and fetch the key, because the robot would not terminate plan execution. The $forward$ action would still be executed, and the robot would be able to move to the key.

In the following sections we introduce the theory behind root cause identification and discuss an algorithm. Afterwards we discuss the new robot control architecture where root cause identification is an integral part. This extended control architecture relies on combining root cause identification and the sense-

plan-act architecture. Then we discuss the software engineering challenges of such systems with respect to development and testing. Finally, we review related research and conclude the paper.

2 Model-Based Reasoning

In this section we introduce the basic concepts of model-based diagnosis (MBD) [3,4] following the definitions of Reiter [18]. The basic idea behind MBD is to use a model of the system directly for identifying the root causes of detected deviations between the observed and the expected behavior. The model specifies the correct behavior of the system. Hence, in the original definition of MBD there is no need for modeling incorrect behavior. This is especially important in cases where the incorrect behavior of components is not known in advance or too expensive to obtain. Since we also want to deal with unknown situations this capability of MBD serves our purpose.

MBD starts with a model, which is called a diagnosis system comprising a system description SD and a set of components $COMP$, where each component may fail. There are two important aspects regarding MBD to mention. First, SD comprises a set of logical sentences that state the model of the components and the structure of the system. In SD also the connections between the components, i.e., the means for communication between components, have to be specified. Second, the component models in SD define the correct behavior of the component. In order to distinguish between the correct and the incorrect behavior MBD makes use of a special predicate AB. $AB(c)$ is true if component c is faulty (or behaves abnormal). The negated predicate $\neg AB(c)$ states that c is working as expected[1].

We illustrate how to write a model for MBD using the fetch key example (Fig. 1), where we model the robot drive partially and in an abstract way. The considered part of the robot drive (see Fig. 3) comprises a motor m connected with a wheel w. The rotation of the wheel is measured using a wheel encoder e. In order to allow for deciding whether the motor is working or not we assume that the current flowing through the motor is also observed by means of a current meter c.

In the following we introduce the logical model SD written in first order logic for our example:

Motor: In case of a correct behavior a particular motor is running in forward direction if there is command for moving forward cmd_fwd. In this case there is a current flowing through the motor. This current can be of nominal value or higher if there is a strong resistance coming from attached components like wheels. Hence, for all components X that are motors, the following logical sentence formalize their behavior:

[1] It is worth noting that it would also be possible to use a predicate NO stating normal operation instead of the negation of AB. However, the AB predicate has been used in almost all MBD papers. Therefore, we also make use of AB in order to introduce the basic MBD theory.

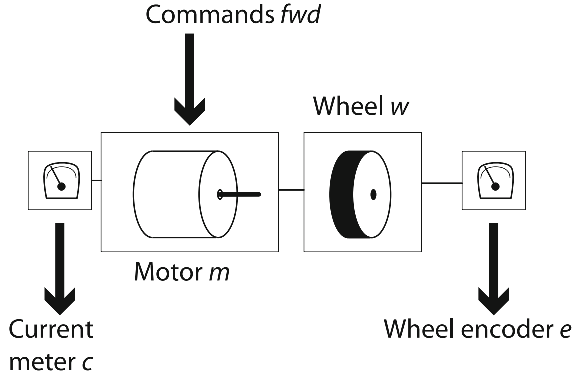


Fig. 3. Schematic of a robot drive

$$\forall X : motor(X) \rightarrow \left(\begin{array}{c} \neg AB(X) \rightarrow (cmd_fwd \rightarrow direction(X, fwd)) \\ direction(X, fwd) \rightarrow torque(X, fwd) \\ direction(X, fwd) \wedge resistance(X) \rightarrow current(X, high) \\ direction(X, fwd) \wedge \neg resistance(X) \rightarrow current(X, nominal) \\ current(X, high) \vee current(X, nominal) \rightarrow direction(X, fwd) \end{array} \right) \quad (1)$$

Wheel: The torque provided via the axle is turned into rotations of the wheel. Only in case of a stuck wheel there is a resistance against the applied torque. But this case is not going to be formalized because we are only interested in the correct behavior. For wheels X we formalize the correct behavior as follows:

$$\forall X : wheel(X) \rightarrow (\neg AB(X) \rightarrow (torque(X, fwd) \rightarrow (rotate(X, fwd) \wedge \neg resistance(X)))) \quad (2)$$

Current Meter: The current meter measures the value of the current. Hence, there is a one to one correspondence of the current flow and the measurement in case of a correct behavior. This behavior of a current meter X can be formalized as follows:

$$\forall X : current_meter(X) \rightarrow \left(\begin{array}{c} \neg AB(X) \rightarrow (current(X, high) \leftrightarrow observed_current(X, high)) \\ \neg AB(X) \rightarrow (current(X, nominal) \leftrightarrow observed_current(X, nominal)) \end{array} \right) \quad (3)$$

Wheel Encoder: The wheel encoder X is providing a frequency only in case of a rotation. For a wheel encoder X we introduce the following logical sentence:

$$\begin{array}{l}\forall X : encoder(X) \rightarrow \\ (\neg AB(X) \rightarrow (rotate(X, fwd) \leftrightarrow frequency(X)))\end{array} \tag{4}$$

What is now missing to complete the model for our example is a description of the structure. We first define the components:

$$motor(m) \wedge wheel(w) \wedge current_meter(c) \wedge encoder(e) \tag{5}$$

Second, we have to define the connections between the components. This can be done using the following logical sentences:

$$\begin{array}{rcl} torque(m, fwd) & \leftrightarrow & torque(w, fwd) \\ resistance(m) & \leftrightarrow & resistance(w) \\ current(m, nominal) & \leftrightarrow & current(c, nominal) \\ current(m, high) & \leftrightarrow & current(c, high) \\ rotate(w, fwd) & \leftrightarrow & rotate(e, fwd) \end{array} \tag{6}$$

The rules stated in Equations (1) – (6) define the structure and behavior of the model SD of our small example. For this example the set of components comprise 4 elements, i.e., $COMP = \{m, c, w, e\}$.

Having now the model $(SD, COMP)$ we are interested in finding root causes. For this purpose we have to state a diagnosis problem.

Definition 1 (Diagnosis problem). *A diagnosis problem is a tuple* $(SD, COMP, OBS)$*, where* SD *is a model,* $COMP$ *a set of components, and* OBS *a set of observations.*

Before discussing solutions to diagnosis problems we should discuss some cases. In the first case assume that the model SD allows for deriving the given observations OBS, or in other words assume that the observations are not in contradiction with the model. In this case, it is reasonable to assume that the system is working as expected. The same holds for the second case where no observations are available, i.e., $OBS = \emptyset$. The third case is the more interesting one, where the observations are in contradiction with SD. In this case, we are interested in finding the root cause for the contradiction. What we have are the components and the assumptions about their correctness or incorrectness. For this purpose we use the negated AB predicate. Hence, a root cause or diagnosis should be assumptions about the logical value of the AB predicate, which lead to a logical sentence that is not in contradiction with the given observations OBS. Formally, we define a diagnosis as follows:

Definition 2 (Diagnosis). *Let* $(SD, COMP, OBS)$ *be a diagnosis problem. A set* $\Delta \subseteq COMP$ *is a diagnosis if and only if* $SD \cup OBS \cup \{\neg AB(X) | X \in COMP \setminus \Delta\} \cup \{AB(X) | X \in \Delta\}$ *is consistent.*

We illustrate this definition using our running example again. Let us assume now that a forward command is sent to the motor, a nominal current is measured, but the wheel encoder is not delivering any frequency output. The diagnosis problem includes SD and $COMP$ described previously and $OBS_1 =$

$\{cmd_fwd, observed_current(m, nominal), \neg frequency(e)\}$. Valid diagnoses are $\{e\}$, $\{w\}$, $\{c, m\}$ but also $\{c, m, e, w\}$. In order to eliminate the latter diagnosis, which states that all components are not working as expected, from the list of diagnoses we introduce the definition of minimal diagnosis.

Definition 3 (Minimal diagnosis). *A diagnosis Δ is a minimal diagnosis if there exists no diagnosis $\Delta' \subset \Delta$.*

From here on we assume that a diagnosis algorithm always returns minimal diagnoses. Hence, the 3 minimal diagnoses $\{e\}$, $\{w\}$, $\{c, m\}$ remain for our running example. This result allows for distinguishing some reasons for a misbehavior. For example, the wheel might be the root cause in case of OBS_1 but not the motor alone. In this case we might prefer small diagnosis and thus reject the fault hypothesis $\{c, m\}$. However, we are not able to distinguish the diagnosis $\{w\}$ from $\{e\}$. In order to solve this challenge there are 3 options: (1) introduce new observations, which might be not possible in case of autonomous systems with a limited number of sensors, (2) use fault probabilities, and (3) extend the model to eliminate diagnoses.

Option (2) can be implemented by assigning a fault probability $p_f(X)$ to each component X. In this case we are able to compute the probability of a diagnosis assuming independence of component faults as follows:

$$p(\Delta) = \prod_{X \in \Delta} p_f(X) \cdot \prod_{X \in COMP \setminus \Delta} (1 - p_f(X)) \tag{7}$$

If we assume that a fault in a wheel is less likely than a fault in the wheel encoder, Equation 7 returns a higher probability value for the diagnosis *wheel encoder* than for the diagnosis *wheel.* It is worth noting that fault probabilities might also change over time. For a theory of adapting fault probabilities using diagnosis information we refer the interested reader to [6].

In option (3) the idea is to eliminate diagnoses by adding knowledge about impossibilities to the system description. Friedrich et al. [10] introduced this concept of *physical impossibilities* and explained the advantage with respect to diagnosis capabilities. For our running example, we might state that it is impossible that the wheel is the reason for observing no frequency in case of nominal current flow through the motor. In other word we state that the physical connection between the motor and the wheel can never break. This is not true in general but might be applicable in special cases. In order to state this impossibility we defined SD' to be SD together with the rule $current(m, nominal) \wedge \neg rotate(w, fwd) \rightarrow \bot$. When using SD' together with OBS_1 only diagnosis $\{e\}$ remains as single diagnosis.

There are many algorithms available for computing diagnosis. Reiter [18] introduces a hitting set based algorithm that makes use of conflicts, which was corrected by Greiner et al. [12]. De Kleer and Williams [6] introduced their general diagnosis engine (GDE). GDE is based on an assumption based truth maintenance system (ATMS) [5]. Later on Nejdl and Fröhlich [11] presented a very fast diagnosis engine that is also based on logic model representations.

Diagnosis algorithms that make use of certain structural properties include [8] and [22].

Note that there is no restriction regarding the modeling language used for model-based reasoning. In this section we make use of first order logic, but also constraint representations [7], or other formalisms might be used. The only requirement is to have a solver that allows for checking consistency. Algorithm 1 introduces a straight forward but not optimal algorithm that computes minimal diagnoses up to a pre-defined size. The algorithm basically checks all subsets of $COMP$ for being a diagnosis. Although there are many improvements possible the algorithm is fast enough if we are interested in smaller diagnoses like single faults only.

Algorithm 1. DIAGNOSIS

Input: A model $(SD, COMP)$, a set of observations OBS, and the maximum size of computed diagnoses $max > 0$. In case max is not given, we assume $max := |COMP|$.
Output: A set of minimal diagnoses accordingly to Definition 3.

```
  {Computes all minimal diagnoses up to the given size.}
  Let DS be the empty set.
  for i:= 0 to max do
     Generate all subset of COMP of size i and store the results in DS'.
     Remove all elements in DS' that are subset of minimal diagnosis stored in DS.
     for all Δ ∈ DS' do
        Call the theorem prover on SD ∪ OBS ∪ {¬AB(X)|X ∈ COMP \ Δ} ∪
        {AB(X)|X ∈ Δ}).
        if the theorem prover call returns an inconsistency then
           Remove Δ from DS'.
        end if
     end for
     Let DS be DS ∪ DS'.
  end for
  return DS
```

3 A Model-Based Control Architecture

In this section, we discuss the extension of the ordinary sense-plan-act control architecture (Fig. 2) to handle cases of internal faults and external events, preventing actions to be executed. In the original architecture a plan is generated that leads from an initial state to a goal state. The state (comprising the sensor information and internal state) is observed and used to select actions based on the plan. The actions are executed using the actuators. This execution might lead to changes in the internal state or in the environment. As discussed in the introduction the original control architecture cannot react on internal faults and hardly interact on external events, preventing actions to be executed and therefore plans to reach their goals.

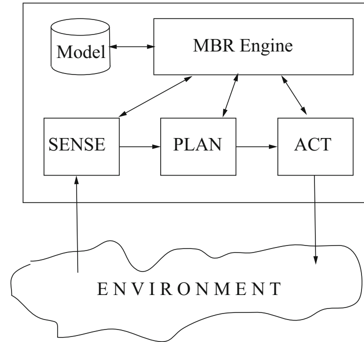


Fig. 4. The adaptive robot control architecture

In order to overcome this problem, we introduce an extended sense-plan-act control architecture, which computes root causes in case of a detected misbehavior. In Figure 4 the extended sense-plan-act control architecture is depicted.

The main control loop of the sense-plan-act architecture remains the same but in some cases, which we discuss later, a model-based reasoner is used to decide the current health status of the autonomous system. The reasoner has access to the internal state of the controller and the sensor information. The health status needs not to be always checked. A check has to be performed only in cases where an action that is decided by the planning module cannot be successfully executed. As already stated, there might be several reasons: (1) there is a fault in the hardware preventing the action to be successfully terminated, or (2) an external event occurred. The latter requires re-planning activities whereas the first reason requires determining the fault in order to adapt the behavior.

Algorithm 2 describes the underlying behavior of the extended sense-plan-act architecture. We assume that the goal state specifies the purpose of an autonomous system. The task of the control architecture is now to find a plan, i.e., a sequence of actions, that allows the system to reach the goal state from the current state, and to execute this plan. This idea originates from STRIPS planning [9] and our algorithm relies on the same input data, i.e., a planning knowledge base M_p comprising a set of actions a where each action has a corresponding set of pre conditions and effects. Moreover, we require a goal state S_G and a diagnosis model $(SD, COMP)$, used in cases of failing actions during plan execution.

In Algorithm 2 we make heavily use of a function SENSE(). This function returns the current state of the autonomous system comprising the internal state and the information obtained from the sensors. We assume that the sensor information is only given when reliable. Hence, in case of a diagnosis indicating a sensor to fail, the information is no longer provided. Although time is not handled explicitly in the algorithm it is worth noting that time exceeds during execution. Therefore, SENSE() represents the state at a discrete point in time only, where measurements and the internal state are observed. We assume that this is done on a regular basis.

Algorithm 2. EXT_PSA_ARCH

Input: A planning model M_p, a diagnosis model $(SD, COMP)$, and the goal state S_G.
Output: Computes and executes a plan from the current state to S_G.

```
Let p := PLAN(M_p,SENSE(),S_G).
while p is not empty do
  Let a be the first action of plan p.
  Remove a from p.
  if pre conditions of a are not fulfilled in SENSE() then
    Let p := PLAN(M_p,SENSE(),S_G).
  else
    Execute a.
    if execution terminates with a failure then
      Let S_Δ be the result of calling DIAGNOSIS(SD,COMP,SENSE()), and Δ be
      the leading diagnosis of S_Δ.
      if Δ indicates a sensor failure only then
        Consider a to be executed without failure and proceed.
      else
        Remove all actions from the planning model that cannot be longer used
        because of diagnosis Δ.
        Let p := PLAN(M_p,SENSE(),S_G).
      end if
    else
      if effects of a are not fulfilled in SENSE() then
        Let p := PLAN(M_p,SENSE(),S_G).
      end if
    end if
  end if
end while
```

The control algorithm starts computing a plan based on the available information, i.e., the current state provided by SENSE() and the planning knowledge based M_p together with the goal state S_G. At the beginning all actions that can be performed by the system are functioning, and are therefore available for planning. Afterwards, the plan is executed by sequentially executing the actions. First, the pre conditions of the current action a are checked. In case that the pre conditions are not fulfilled in the current state, the action cannot be performed. This might happens due to an external event. As a consequence, re-planning has to be performed and plan execution starts again using the new plan.

If the pre conditions of an action are fulfilled, the action is executed. This execution might be terminated returning a failure. In this case diagnosis has to be performed that returns diagnoses from which a leading diagnosis can be obtained. A leading diagnosis in our case might be the smallest diagnosis or the one with the highest fault probability. For simplicity we do not handle the case of multiple diagnoses that cannot be distinguished with the available information. If such case occurs, measures for distinguishing diagnoses have to be performed, like providing testing procedures. For example, Wotawa et al. [25] introduce

distinguishing test cases for solving such problems. Hence, we assume that we can always determine a leading diagnosis. This leading diagnosis is used to either remove actions that cannot be performed anymore, or to assume that some sensor data is no longer reliable. In the latter case there is no need to apply re-planning. Instead the action is forced to be executed and the procedure continues.

The last possibility for a fault occurring during execution is that the effects of an action are not visible. In this case we again perform re-planning, which might lead to the case where the current action is re-executed again. Note that this might lead to a situation where the robot is executing an action again and again due to a sensor failure or an external event. Therefore, in the implementation the repetition of executions of the same actions should be tracked and handled appropriately. For example, a diagnosis step might also be performed.

The extended sense-plan-act control architecture provides adaptation due to internal faults and external events. Depending on the used models it allows for distinguishing sensor and actuator faults. It might also be used for handling faults in software providing a model of the behavior. Of course the implementation of the control loop and the diagnosis module have to be correct and cannot be corrected using the proposed approach.

4 Testing Model-Based Systems

One advantage of the model-based reasoning approach is that the underlying implementation, i.e., the diagnosis module and the control architecture, can be easily re-used. The only changes necessary are changes of the underlying models. Therefore, we focus on challenges of testing models instead of testing the implementation of the different modules that together contribute to the overall system's architecture. Testing models especially when they are written in logic or any other relational language is rather similar to ordinary software testing [1]. In both cases we are interested in comparing the behavior imposed by a system with the expected behavior. However, there are some differences and challenges like answering the question when to stop testing? Usually, in software engineering practice there are some criteria like mutations scores or coverage, e.g., statement or branch coverage, a test suite has to fulfill. In case of testing models such criteria can hardly be applied directly and new criteria have to be developed.

We start with discussing testing the diagnosis model. What we have to ensure is that the model allows for deriving all correct diagnoses for any given set of observations. Moreover, the model should never allow deriving diagnoses that are not correct. So what we want to show is whether DIAGNOSIS(SD,$COMP$,OBS) $= DS_{expected}$ holds for all possible OBS or not. This is of course not feasible even in the case of a finite number of predicates that might be used in OBS. Therefore, in testing diagnosis models we have to restrict the number of tests. Formally, we introduce a test suite for a model $(SD, COMP)$ as follows:

Definition 4 (Test, test suite). *Let* $(SD, COMP)$ *be a diagnosis model and* PO *the set of possible observations. A tuple* (OBS, DS, max)*, where* OBS *is*

a set of observations, DS a set of diagnoses with cardinality less or equal to max, is a test for the diagnosis model if $OBS \subseteq PO$ *and* $SD \cup OBS \not\models \bot$ *or if* $OBS \subseteq PO$, $SD \cup OBS \models \bot$ *and* $DS = \emptyset$ *. A test suite for the diagnosis model is a set of tests.*

Note that in the definition $\models$ stands for the logical consequence, and $\bot$ for the contradiction. In the above definition there is no restriction on DS with the exception of the case, where the observations directly are in contradiction with the model. In this case no diagnosis is allowed to be computed. This fact follows directly from the definition of diagnosis in MBD.

We are now able to specify soundness and completeness of a model with respect to the given test suite. In particular, we might be interested whether a given model delivers all correct diagnoses. This can be ensured when having sound and complete models. A model is sound if only correct diagnoses can be obtained from the model. A model is complete if all diagnoses can be derived from the model and the given observations.

Definition 5 (Soundness, completeness). *Let* $(SD, COMP)$ *be a diagnosis model and* TS *its non empty test suite.* $(SD, COMP)$ *is sound with respect to* TS *iff for all* $(OBS, DS, max) \in TS$: *DIAGNOSIS(SD,COMP,OBS,max)* $\subseteq DS$. $(SD, COMP)$ *is complete with respect to* TS *iff for all* $(OBS, DS) \in TS$: *DIAGNOSIS(SD,COMP,OBS,max)* $\supseteq DS$.

We further say that a diagnosis model is correct with respect to a test suite iff it is sound and complete.

After defining test suites and their impact on the model under test, we are interested in generating test suites that obey a reasonable criterion. Although we know that *"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."*[2] generating tests that at least allows us to check correctness for some important cases is essential. Hence, we have to ask ourselves what are the important cases for diagnosis? When considering the practical application of diagnosis, correctly identifying single fault might be enough. Double and triple faults might occur but can hardly be distinguished. Therefore, there would be a strong need for handling a huge amount of diagnoses concurrently, which increases complexity. Hence, classifying test suites regarding their capabilities of testing diagnosis results up to a certain cardinality, seems to be a good choice.

We now generalize the concept of classifying test suites with respect to the checking diagnoses up to a certain cardinality.

Definition 6 (n-cardinality criteria). *A test suite* TS *for a diagnosis model* $(SD, COMP)$ *fulfills the* $n-$*cardinality criteria iff* $\forall \Delta \subseteq 2^{COMP} \wedge |\Delta| \leq n : \exists (OBS, DS, max) \in TS : (\Delta \in DS \vee \exists \Delta' \in DS : \Delta \supset \Delta')$.

Obviously in this definition a test suite that fulfills 2-cardinality also fulfills 1-cardinality. Therefore, when constructing a test suite, starting with 1-cardinality

[2] from Edsger Wybe Dijkstra, The Humble Programmer, ACM Turing Lecture 1972.

and than extending the test suite to fulfill the higher cardinality criteria, seems to be a good procedure. It is worth noting that even constructing 1-cardinality test suites may not be easy because specifying all diagnoses of size 1 for given observations can be a hard problem. Hence, in practice it might be a good option to iteratively construct test cases. First, start with observations that must lead to some diagnosis. Call the DIAGNOSIS algorithm for computing all diagnosis up to a given size. If the intended diagnoses are part of the result, check the remaining diagnoses (if there are any). Remove all diagnoses that are unexpected. Add diagnoses that are missing and generate the new test case.

Besides testing the diagnosis model the planning model and the interaction of the planning module and the diagnosis module has to be tested. The planning model might be tested similarly to the diagnosis model. The initial and the goal state have to be specified, and possible plans have to be provided. If the planner computes the same plans, the model has to be correct with respect to the test case. Again certain corner cases should be specified and tested in order to ensure at least partial correctness.

The interaction between the planner and the diagnosis engine can be tested by stating more complex scenarios. The integration test has to specify a scenario where a fault occurs, which has to be detected by the diagnosis algorithm. If the implementation together with the model fulfill the scenarios the model might be assumed to be correct because the system behaves like expected. However, because of complexity it is impossible to check all scenarios. Again only some important cases can be used for evaluation purposes.

Testing model-based applications basically means testing the developed models. Because of the complexity of models and underlying domains it is a painstaking task to generate larger sets of test cases. Moreover, the quality of test suites can hardly be quantified. In case of the model-based diagnosis module we are able to quantify test suites with respect to the cardinality of diagnoses handled in a certain case. What remains to do is to work on supporting test case generation for model-based systems. Even for diagnosis this seems not to be an easy task and requires an iterative procedure.

5 Related Research

The application of MBD in the context of autonomous mobile systems is not new. To our knowledge Williams and colleagues [24,17] were the first introducing a system that automatically adapts to internal faults in order to reach its objectives. The MBD system had been tested in real-world conditions as part of the control system of the NASA deep space one space probe[3]. The focus of the approach was on hardware faults. Hence, intelligent interactions in case of external events were not considered. Moreover, the space probe itself had no intended capabilities of changing the external world. This is different to the mobile robotics domain where actions have an impact on the environment.

[3] See `http://nmp.nasa.gov/ds1/`

Approaches for applying MBD to the domain of robotics include [2], [15], and [16] where the focus lies more in diagnosing interactions between many mobile robots that should work together in order to reach a specified goal. Steinbauer and colleagues [19] introduced a system that is capable of dealing with software faults in mobile robot control systems. The approach is based on a very simplified structural model of the control software and makes use of monitoring procedures for detecting a misbehavior. The structural model is used to extract pieces of software, i.e., modules, that should be restarted in order to overcome the detected problems. The approach of Steinbauer and colleagues does not make use of a behavior model and does not consider the case of external events.

In [14] the authors describe a MBD approach for handling faults in robot drives. In particular the authors claim that they can adapt the kinematics of the robot in case of faults in the drive. In the paper the focus is on how to adapt the kinematics. The whole system's architecture is explained but not in detail. The adaption of high level control is also not given in detail. However, we borrow the idea of using diagnosis information for changing high-level control from [14].

The idea of adapting plans based on diagnosis knowledge gained during execution was outlined in [23] and [21]. The first paper deals in particular with dependent failures, which might occur in robotics systems. In both papers the authors make use of repair in case of misbehavior to ensure robustness. In this paper we formally introduce the system's architecture and also discuss the challenge of verifying such systems by means of model testing.

The work most closely to our work described in this paper is [13]. There the authors introduce a MBD approach for mobile autonomous systems that is capable to handle internal faults as well as external events. The approach of Gspandl et al. makes use of a Reiter's situation calculus and is characterized by combining control and diagnosis knowledge in a uniform framework. In contrast to this work our approach separates planning and control from diagnosis. Moreover, we also tackle the problem of testing such systems, which has not been covered before.

Steinbauer and Wotawa [20] discussed the evaluation of adaptive mobile and autonomous robots. There the objective is to find an evaluation testbed that allows for comparing different solutions. The paper does not deal with testing such systems during development. Instead the paper focuses on the validation of intelligent and adaptive robots with respect to their robustness in practical situations.

6 Conclusion

In this paper we introduced a control architecture that enables adaptive control of autonomous systems. The control architecture integrates the sense-plan-act architecture and a model-based diagnosis engine. As a consequence both external events and internal faults can be treated in a smart way. Internal faults are handled by the diagnosis engine directly. External events cause re-planning activities. An advantage of the proposed system is also the implicit software re-use.

The implementation of the approach could be used in many applications. Only the underlying models have to be adapted or changed.

From the software engineering point of view, there is a focus switch from coding to modeling. Or in other words, model-based development requires writing models instead of more traditional source code. Consequently, testing has to focus on model testing. Unfortunately, testing models is not an easy task. In the paper we outline some basic definitions of testing models for diagnosis and briefly discuss an iterative approach for generating test suites from models.

Although, authors have shown in their papers that the model-based reasoning approach really allows for generating adaptive autonomous systems, there is almost no work on developing and testing such systems. In future work we will discuss this issue in much more detail. It is also worth mentioning that the proposed approach requires a certain degree of redundancy. This redundancy is different from the redundancy used to ensure robustness and reliability of systems, where components are typically available many times. In case of MBD there is a need for an overlap of functionality, e.g., sensor input, in the system that is maybe provided by different sensors. Developing guidelines for constructing systems in order to fulfill the requirement of partial redundancy is also an open research issue.

Acknowledgement. The work presented in the paper has been funded by the Austrian Science Fund (FWF) under contract number P22690.

References

1. Beizer, B.: Software Testing Techniques. Van Nostrand Reinhold (1990)
2. Daigle, M., Koutsoukos, X., Biswas, G.: Distributed diagnosis of coupled mobile robots. In: IEEE International Conference on Robotics and Automation, pp. 3787–3794 (2006)
3. Davis, R.: Diagnostic reasoning based on structure and behavior. Artificial Intelligence 24, 347–410 (1984)
4. Davis, R., Hamscher, W.: Model-based reasoning: Troubleshooting. In: Shrobe, H.E. (ed.) Exploring Artificial Intelligence, ch.8, pp. 297–346. Morgan Kaufmann (1988)
5. de Kleer, J.: An assumption-based TMS. Artificial Intelligence 28, 127–162 (1986)
6. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artificial Intelligence 32(1), 97–130 (1987)
7. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
8. El Fattah, Y., Dechter, R.: Diagnosing tree-decomposable circuits. In: Proceedings 14th International Joint Conf. on Artificial Intelligence, pp. 1742–1748 (1995)
9. Fikes, R.E., Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, 189–208 (1971)
10. Friedrich, G., Gottlob, G., Nejdl, W.: Physical impossibility instead of fault models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Boston, pp. 331–336 (August 1990); also appears in Readings in Model-Based Diagnosis. Morgan Kaufmann (1992)

11. Fröhlich, P., Nejdl, W.: A Static Model-Based Engine for Model-Based Reasoning. In: Proceedings 15th International Joint Conf. on Artificial Intelligence, Nagoya, Japan (August 1997)
12. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence 41(1), 79–88 (1989)
13. Gspandl, S., Pill, I.H., Reip, M., Steinbauer, G., Ferrein, A.: Belief management for high-level robot programs. In: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI (2011)
14. Hofbaur, M., Köb, J., Steinbauer, G., Wotawa, F.: Improving robustness of mobile robots using model-based reasoning. Journal of Intelligent & Robotic Systems 48(1), 37–54 (2007)
15. Kalech, M., Kaminka, G.A.: On the design of coordination diagnosis algorithms for teams of situated agents. Artificial Intelligence 171(8-9), 491–513 (2007)
16. Micalizio, R., Torasso, P., Torta, G.: On-line monitoring and diagnosis of a team of service robots: A model-based approach. AI Communications 19(4), 313–340 (2006)
17. Rajan, K., Bernard, D.E., Dorais, G., Gamble, E.B., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P.P., Rouquette, N.F., Smith, B.D., Taylor, W., Tung, Y.: Remote agent: An autonomous control system for the new millennium. In: 14th European Conference on Artificial Intelligence (ECAI), pp. 726–730 (2000)
18. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
19. Steinbauer, G., Mörth, M., Wotawa, F.: Real-time diagnosis and repair of faults of robot control software. In: RoboCup International Symposium, pp. 13–23 (2005)
20. Steinbauer, G., Wotawa, F.: Evaluating the robustness of the perception-decision-execution cycle of autonomous robots. In: Proceedings of the ICAR Workshop on Performance Measures for Quantifying Safe and Reliable Operation of Professional Service Robots in Unstructured, Dynamic Environments (2011)
21. Steinbauer, G., Wotawa, F.: Robust plan execution using model-based reasoning. Advanced Robotics 23(10), 1315–1326 (2009)
22. Stumptner, M., Wotawa, F.: Diagnosing Tree-Structured Systems. In: Proceedings 15th International Joint Conf. on Artificial Intelligence, Nagoya, Japan (1997)
23. Weber, J., Wotawa, F.: Diagnosis and repair of dependent failures in the control system of a mobile autonomous robot. Applied intelligence (2008)
24. Williams, B.C., Pandurang Nayak, P.: Immobile robots – ai in the new millennium. AI Magazine, 16–35 (1996)
25. Wotawa, F., Nica, M., Aichernig, B.K.: Generating distinguishing tests using the minion constraint solver. In: CSTVA 2010: Proceedings of the 2nd Workshop on Constraints for Testing, Verification and Analysis. IEEE (2010)

Representing Knowledge in Robotic Systems with KnowLang

Emil Vassev and Mike Hinchey

Lero—the Irish Software Engineering Research Centre,
University of Limerick, Limerick, Ireland
{Emil.Vassev,Mike.Hinchey}@lero.ie

Abstract. Building intelligent robotic systems is both stirring and extremely challenging. Researchers have realized that robot intelligence can be achieved with a logical approach, but still AI struggles to connect that abstract logic with real-world meanings. This paper presents KnowLang, a new formal language for knowledge representation in a special class of intelligent robotic systems termed ASCENS. Autonomic Service-Component Ensembles (ASCENS) are multi-agent systems formed as mobile, intelligent and open-ended swarms of special autonomic service components capable of local and distributed reasoning. Such components encapsulate rules, constraints and mechanisms for self-adaptation and acquire and process knowledge about themselves, other service components and their environment. In this paper,a brief KnowLang case study of knowledge representation for a robotic system is presented.

Keywords: knowledge representation, intelligent robotic systems, formal approach, robot ontology, ASCENS.

1 Introduction

Nowadays one of the most intriguing challenges in IT is the challenge of building intelligent robots. Apart from the complex mechanisms and electronics, building robots is about the challenge of interfacing with a dynamic and unpredictable world, which requires the presence of intelligence. Robotic artificial intelligence (AI) mainly excels at formal logic, which allows it,for example, to find the right chess move from hundreds of previous games. According to Matt Berlin, an AI researcher with MIT's Media Lab,"*People realized at some point that you can only get so far with a logical approach*". The problem is that AI struggles to connect that abstract logic with real-world meanings, which willl give a robot the necessary knowledge to become intelligent.

The basic building block of intelligence in robotic systems is *data* [1], which takes the form of measures and representations of the internal and external worlds of a robot, e.g., raw facts and numbers. When regarded in a specific context (domain of interest), data can be assigned relevant meaning to become *information*. Consecutively, knowledge is a specific interpretation of information, i.e., knowledge is created and organized by flows of information interpreted and shaped by the

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 91–105, 2012.
© Springer-Verlag Berlin Heidelberg 2012

intelligent system. Here the most intriguing question is how to represent the data and what mechanisms and algorithms are needed to derive knowledge from it.

In this paper, we present our approachto knowledge representation for a particular class of intelligent robotic systems termed Autonomic Service-Component Ensembles (ASCENS) [2]. Our assumption is that knowledge representation can be regarded as a *formalspecification* of knowledge data reflecting the robot's understanding about itself and its surrounding world. To specify a knowledge representation in ASCENS systems, we are currently developing a special formal language termed as KnowLang. In this paper, we present KnowLang in terms of special specification tiersand parameterization necessary to cover the specification of robot's knowledge domains and reasoning primitives.We use KnowLang in a simple knowledge representation for a robotic case study.

The rest of this paper is organized as follows: Section 2 introduces the notion of knowledge together with common knowledge representation techniques and inference engines. In Section 3, we briefly present ASCENS and our approach to knowledge representation for ASCENS systems. In Section 4, we formally present KnowLang, our target language for specifying knowledge in ASCENS systems. Section 5 presents a knowledge-representation case study and finally in Section 6, we provide brief concluding remarks and a summary of our future research goals.

2 Background

Conceptually, knowledge can be regarded as a large complex aggregation [3] composed of constituent parts representing knowledge of different kind. Each kind of knowledge may be used to derive knowledge models of specific domains of interest. For example, in [3] the following kinds of knowledge are considered:

- *domain knowledge* – refers to the application domain facts, theories, and heuristics;
- *control knowledge* – describes problem-solving strategies, functional models, etc.;
- *explanatory knowledge* – defines rules and explanations of the system's reasoning process, as well as the way they are generated;
- *system knowledge* – describes data contents and structure, pointers to the implementation of useful algorithms needed to process both data and knowledge, etc. System knowledge may also define user models and strategies for communication with users.

Moreover, being considered as essential system and environment information, knowledge may be classified as 1) *internal knowledge* – knowledge about the system itself; and 2) *external knowledge* – knowledge about the system environment. Another knowledge classification could consider *a priori knowledge* (knowledge initially given to a system) and *experience knowledge* (knowledge gained from analysis of tasks performed during the lifetime of a system).

There are different knowledge representation techniques that might be used to represent different kinds of knowledge and it is our responsibility to pick up or create the technique which best suits our needs.In general, to build a knowledge model we

need specific *knowledge elements*. The latter may be primitives such as *frames*, *rules*, *logicalexpressions*, etc. Knowledge primitives might be combined together to represent more complex knowledge elements. A knowledge model may classify knowledge elements by type, and group those of the same type into collections. Typical knowledge representation techniques are *rules*, *frames*, *semanticnetworks*, *concept diagrams*, *ontologies* and *logics* [4, 5]. Actually logics are used to formalise the knowledge representation techniques, which gives them a precise semantics. *Knowledge-based systems* integrate knowledge via knowledge representation techniques to build a computational model of some domain of interest in which symbols serve as *knowledge surrogates* for real world domain artefacts, such as physical objects, events, relationships, etc. The *domain of interest* can cover any part of the real world or any hypothetical system about which one desires to represent knowledge for computational purposes. Computations over represented knowledge are done by the so-called *inference engine* (or *inferential engine*) that acts on the knowledge facts to produce other facts that may need to be added to the knowledge base (KB). For example, if the KB contains *rules*, the inference engine may chain them either forward (e.g., for forecast) or backward (e.g., for diagnosis). The inference engines are logic-based, e.g., First Order Logic (FOL) and Description Logics (DL) [5, 6].

One way to implement inference is by using algorithms from automated deduction dedicated to FOL, such as *theorem proving* and *model building*. Theorem proving can help in finding contradictions or checking for new information. Finite model building can be seen as a complementary inference task to theorem proving, and it often makes sense to use both in parallel. Some common FOL-based inference engines are VAMPIRE [7], SPASS [8], and the E theorem prover [9]. The problem with FOL-based inference is that the logical entailment for FOL is *semi-decidable*, which means that if the desired conclusion follows from the premises then *eventually* resolution refutation will find a contradiction. As a result, queries often unavoidably do not terminate.Inference engines based on Description Logics (DLs) (e.g., Racer [10], DLDB [11], etc.) are extremely powerful when reasoning about *taxonomic knowledge*, since they can discover hidden *subsumptionrelationships* amongst classes. However, their expressive power is restricted in order to reduce the computational complexity and to guarantee the *decidability* (DLs are decidable) of their deductive algorithms. Consequently, this restriction prevents taxonomic reasoning from being widely applicable to heterogeneous domains (e.g. integer and rational numbers, strings) in practice.

3 ASCENS and ASCENS Knowledge Base

ASCENS is an FP7 (Seventh Framework Program) [12] project targeting the development of a coherent and integrated set of methods and tools providing a comprehensive development approach to developing *ensembles* (or swarms) of intelligent, autonomous, self-aware and adaptive *service components* (SC). Note that it is of major importance for an ASCENS system to acquire and structure comprehensive knowledge in such a way that it can be effectively and efficiently processed, so such a system becomes aware of itself and its environment. Our initial

research on knowledge representation for ASCENS systems [4, 13] concluded that a SC should have structured knowledge addressing the SC's structure and behaviour, the SC Ensemble's (SCE) structure and behaviour, the environment entities and behaviour and situations where that SC or the entire SCE might end up in. Based on these considerations, we defined four *knowledge domains* in ASCENS [4]:

- *SC knowledge* – knowledge about robot's internal configuration, resource usage, content, behaviour, services, goals, communication ports, actions, events, metrics, etc.;
- *SCE knowledge* – knowledge about the whole system, e.g., architecture topology, structure, system-level goals and services, behaviour, communication links, public interfaces, system-level events, actions, etc.;
- *environment knowledge* – parameters and properties of the operational environment, e.g., external systems, concepts, objects, external communication interfaces, integration with other systems, etc.;
- *situational knowledge* – specific situations, involving one or more SCs and eventually the environment.

These knowledge domains are represented by four distinct *knowledge corpuses* — SC Knowledge Corpus, SCE Knowledge Corpus, Environment Knowledge Corpus and Situational Knowledge Corpus. Each knowledge corpus is structured into a special *domain-specific ontology* [14] and a *logical framework*. The domain-specific ontology gives a *formal* and *declarativerepresentation* of the knowledge domain in terms of explicitly described domain concepts, individuals (or objects) and the relationships between those concepts/individuals. The *logical framework* helps to realize the explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers, and identity. The logical framework provides additional computational structures to determine the logical foundations helping a SC reason and infer knowledge.

All four ASCENS knowledge corpuses together form the ASCENS Knowledge Base (AKB). The AKB is a sort of *knowledge database* where knowledge is stored, retrieved and updated. In addition to the knowledge corpuses, the AKB implies a *knowledge-operating mechanism* providing for knowledge *storing*, *updating* and *retrieval/querying*. Ideally, we can think of an AKB as a black box whose interface consists of two methods called TELL and ASK. TELL is used to add new sentences to the knowledge base and ASK can be used to query information. Both methods may involve *knowledge inference*, which requires that the AKB is equipped with a special *inference engine* (or multiple, co-existing inference engines) that reasons about the information in the AKB for the ultimate purpose of formulating new conclusions, i.e., inferring new knowledge.

4 KnowLang

KnowLangis a formal language providing a comprehensive specification model addressing all the aspects of an ASCENS Knowledge Corpus and providing formalism sufficient to specify the AKB operators and theories helping the AKB inference mechanism. The complexity of the problem necessitates the use of a

specification model where knowledge can be presented at *different levels of depth of meaning*. Thus, KnowLang imposes a multi-tier specification model (see Figure 1), where we specify the ASCENS knowledge corpuses, KB operators and inference primitives at different hierarchically organized *knowledgetiers*.

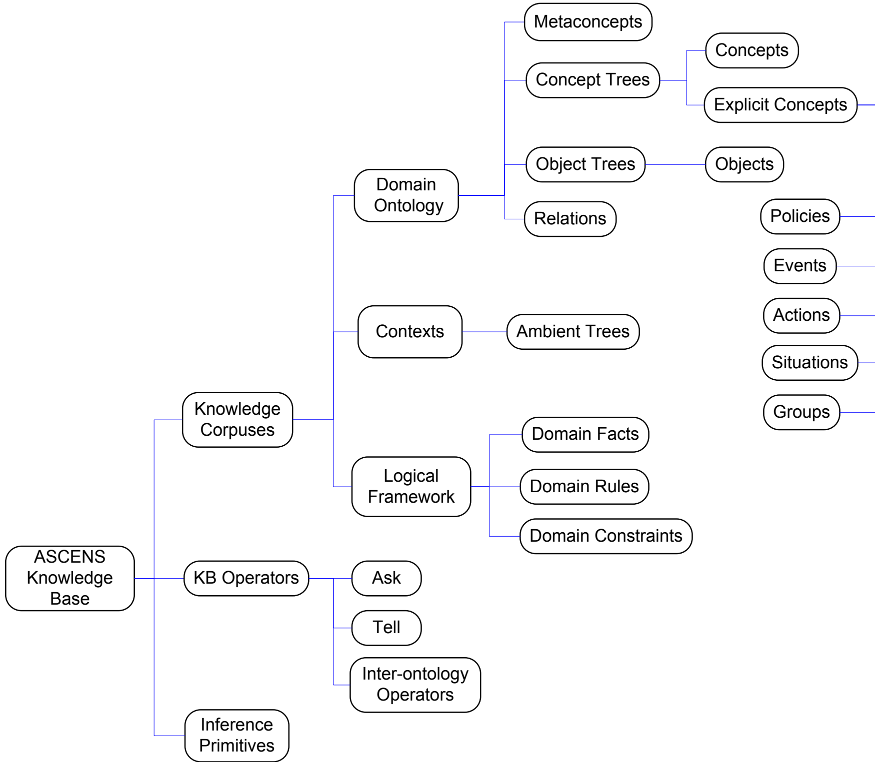


Fig. 1. KnowLang Multi-tier Specification Model

Definitions 1 through 49 (see the definitions following Figure 1) outline a formal representation of the KnowLang specification model. As shown in Definition 1, an ASCENS Knowledge Base (AKB) is a tuple of three main knowledge components - *knowledge corpus* (Kc), *KBoperators* (Op) and *inference primitives* (Ip). A Kc is a tuple of three knowledge components – *ontologies* (O), contexts (Cx) and *logical framework* (Lf) (see Definition 2).

Further, an ASCENS ontology is composed of hierarchically organized sets of *meta-concepts* (Cm), *concept trees* (Ct), *object trees* (Ot) and *relations* (R) (see Definition 4). Meta-concepts (Cm) provide a context-oriented *interpretation* (i) (see Definition 6) of concepts.

Concept trees (Ct) consist of semantically related *concepts* (C) and/or explicit *concepts* (Ce). Every *concept tree* (ct) has a *root concept* (tr) because the architecture ultimately must reference a single concept that is the connection point to concepts that are outside the concept tree. A root concept may *optionally* inherit a

meta-concept, which is denoted $[tr \twoheadrightarrow cm]$(see Definition 8). The square brackets "[]" state for "optional" and "↠" is *inherits* relation. Every concept has a set of *properties* (P) and optional sets of functionalities (F), *parent concepts* (Pr) and *children concepts* (Ch) (see Definition 10).

Explicit concepts are concepts that **must** be presented in the knowledge representation of an ASCENS system. Explicit concepts are mainly intended to support 1) the autonomic behaviour of the SCs; and 2) distributed reasoning and knowledge sharing among the SCs. These concepts might be *policies* (Π), *events* (E), *actions* (A), *situations* (Si) and *groups* (Gr) (see Definition 13).

A policy has a *goal* (g) and *policyconditions* (N_π) *mapped* to *policyactions*(A_π), where the evaluation of N_π may imply the evaluation of actions (denoted with $(N_\pi \longrightarrow A_\pi)$ (see Definition 15). A *condition* is a Boolean function over ontology (see Definition 17). Note that the policy conditions may be expressed with *policy events*.

FORMAL REPRESENTATION OF KNOWLANG

$Kb := \{Kc,\ Op,\ Ip\}$	(*ASCENS Knowledge Base*)	(1)
$Kc := \{O,\ Cx,\ Lf\}$	(*ASCENS Knowledge Corpus*)	(2)

ASCENS ONTOLOGIES

$O := \{o_{sc}, o_{sce}, o_{env}, o_{si}\}$	(*ASCENS Ontologies*)	(3)
$o := \{Cm,\ Ct, Ot, R\}$	(*ASCENS Ontology*)	(4)
$o \in O$		
$Cm := \{cm_0, cm_1, \ldots, cm_n\}$	(*Meta-concepts*)	(5)
$cm := \{[cx],\ i\}$	(*Meta-concept, cx – Context*)	(6)
$i \in Icx$	(i *– Interpretation*)	
$Ct := \{ct_0,\ ct_1, \ldots, ct_n\}$	(*Concept Trees*)	(7)
$ct := \{tr,\ C,\ [Ce]\}$	(*Concept Tree*)	(8)
$tr \in (C \cup Ce), [tr \twoheadrightarrow cm]$	(*tr – Tree Root*)	
$C := \{c_0,\ c_1, \ldots, c_n\}$	(*Concepts*)	(9)
$c := \{P, [F], [Pr], [Ch]\}$	(*Concept*)	(10)
$Pr \subset (C \cup Ce),\ c \twoheadrightarrow Pr$	(*Pr – Parents*)	
$Ch \subset (C \cup Ce),\ c \twoheadleftarrow Ch$	(*Ch – Children*)	
$P := \{p_0, p_1, \ldots, p_n\}$	(*Properties*)	(11)
$F := \{f_0, f_1, \ldots, f_n\}$	(*Functionalities*)	(12)
$Ce := \{\Pi, E, A, Si, Gr\}$	(*Explicit Concepts*)	(13)
$\Pi := \{\pi_0, \pi_1, \ldots, \pi_n\}$	(*Policies*)	(14)
$\pi := \{g,\ N_\pi,\ A_\pi,\ map(N_\pi,\ A_\pi)\}$	(*Policy*)	(15)
$A_\pi \subset A, N_\pi \longrightarrow A_\pi$	$(A_\pi - Policy\ Actions)$	
$E_\pi \subset E, E_\pi \subset N_\pi$	$(E_\pi - Policy\ Events)$	
$N_\pi := \{n_0, n_1, \ldots, n_n\}$	$(Policy\ Conditions)$	(16)
$n := bf(O)$	(*Condition – Boolean Statement*)	(17)
$g := (s \Longrightarrow s')$	(*Goal*)	(18)
$s := \langle Tell \rhd ob.P\rangle \mid \langle Tell \rhd \{ob_0.P, ob_1.P, \ldots, ob_n.P\}\rangle \mid \langle Tell \rhd E_s\rangle$	(*State*)	(19)
$E_s \subset E$	(E_s *–StateEvents*)	
$E := \{e_0, e_1, \ldots, e_n\}$	(*Events*)	(20)

$A := \{a_0, a_1, \ldots, a_n\}$	(*Actions*)	(21)
$Si := \{si_0, si_1, \ldots, si_n\}$	(*Situations*)	(22)
$si := \{s, A_{si}^{\leftarrow}, [E_{si}^{\leftarrow}], A_{si}\}$	(*Situation*)	(23)
$A_{si}^{\leftarrow} \subset A$	$(A_{si}^{\leftarrow} - ExecutedActions)$	
$A_{si} \subset A$	$(A_{si} - PossibleActions)$	
$E_{si}^{\leftarrow} \subset E$	$(E_{si} - SituationEvents)$	
$Gr := \{gr_0, gr_1, \ldots, gr_n\}$	(*Groups*)	(24)
$gr := \{Ob_{gr}, R_{gr}\}$	(*Group*)	(25)
$Ob_{gr} \subset Ob$	$(Ob_{gr} - Group\ Objects, Ob - Objects)$	
$R_{gr} \subset R$	$(R_{gr} - Group\ Relations)$	
$Ot := \{ot_0, ot_1, \ldots, ot_n\}$	(*Object Trees*)	(26)
$ot := \{ob, [Pb]\}$	(*Object Tree*)	(27)
$ob := \{instof(c), P\}$	(*Object*)	(28)
$ob \in Ob$		
$Pb := \{ob_0, ob_1, \ldots, ob_n\}$	(*Object Properties*)	(29)
$Pb \subset P$		
$R := \{r_0, r_1, \ldots, r_n\}$	(*Relations*)	(30)
$r := \{c_k, rn, c_n\} \mid \{ob_k, rn, ob_n\}$	(*Relation, rn – relation name*)	(31)

A *goal* is a desirable transition from a *state* to another *state* (denoted with $s \Rightarrow s'$) (see Definition 18). The system may occupy a state (s) when the *properties* of an object are updated (denoted with $Tell \rhd ob.P$), the *properties* of a set of objects get updated, or some events have occurred in the system or in the environment (denoted with $Tell \rhd E_s$) (see Definition 19). Note that *Tell* is a KB Operator involving knowledge inference (see Definition 46).

A *situation* is expressed with a state (s), a *history of actions* ($A_{si}^{\leftarrow}$) (actions executed to get to state s), *actions*A_{si} that can be performed from state s and an optional *history ofevents*$E_{si}^{\leftarrow}$ eventually occurred to get to state s (see Definition 23).

A *group* involves objects related to each other through a distinct set of relations (see Definition 25). Note that groups are an explicit concept intended to (but not restricted) represent knowledge about the SCE structure topology.

Object trees (Ot) are conceptualization of how objects existing in the world of interest are related to each other. The relationships are based on the principle that objects have properties, where sometimes the value of a property is another object, which in turn also has properties. Such properties are termed as *object properties* (Pb). An object tree consists of a root object (ob) and an optional set of object properties (Pb) (see Definition 27). An *object* (ob) has a set of *properties* (P) including object properties (Pb) and is an instance of a concept (denoted as $instof(c)$ - see Definition 28).

Relations connect two concepts or two objects. Note that we consider *binary relations* only.

ASCENS CONTEXTS

$Cx := \{cx_0, cx_1, \ldots, cx_n\}$	(*Contexts*)	(32)
$cx := \{At, [Icx]\}$	(*Context*)	(33)
$At := \{at_0, at_1, \ldots, at_n\}$	(*AmbientTrees*)	(34)

$$at := \{ct,\ Ca, [i]\} \qquad (AmbientTree) \qquad (35)$$
$$ct \in Ct \qquad (Concept\ Tree\ described\ by\ anontology)$$
$$Ca \subset C \qquad (Ca - Ambient\ Concepts)$$
$$i \in Icx \qquad (i - Ambient\ Tree\ Interpretation)$$
$$Icx := \{i_0,\ i_1, \ldots, i_n\} \qquad (Context\ Interpretations) \qquad (36)$$

Contexts are intended to extract the relevant knowledge from an ontology. Moreover, contexts carry interpretation for some of the meta-concepts (see Definition 6), which may lead to a new interpretation of the *descendant concepts* (derived from a meta-concept – see Definition 8). We consider a very broad notion of context, e.g., the environment in a fraction of time or a generic situation such as currently-ongoing important system function. Thus, a context must emphasize the key concepts in an ontology, which helps the inference mechanism narrow the domain knowledge (domain ontology) by exploring the concept trees down to the emphasized key concepts only. Thus, depending on the context, some low-level concepts might be subsumed by their upper-level parent concepts, just because the former are not relevant for that very context. For example, a robot wheel can be considered as a thing or as an important part of the robot's motion system. As a result, the context interpretation of knowledge will help the system deal with "clean" knowledge and the reasoning shall be more efficient. A context (cx) consists of *ambient trees* (At) and optional *contextinterpretations* (Icx) (see Definition 33). An *ambient tree* (at) consists of a real concept tree (ct) described by an ASCENS ontology, *ambient concepts* (Ca) part of the concept tree and optional *contextinterpretation* (i).

ASCENS LOGICAL FRAMEWORK

$$Lf := \{Fa, Rl, Ct\} \qquad (ASCENS\ Logical\ Framework) \qquad (37)$$
$$Fa := \{fa_0, fa_1, \ldots, fa_n\} \qquad (Facts) \qquad (38)$$
$$fa := bf(O) \longrightarrow \mathbf{T} \qquad (Fact - True\ statement\ over\ ontology) \qquad (39)$$
$$Rl := \{rl_0, rl_1, \ldots, rl_n\} \qquad (Rules) \qquad (40)$$
$$rl := if\ \ fa_1\ then\ fa_2\ [else\ fa_3] \qquad (Rule) \qquad (41)$$
$$Ct := \{ct_0, ct_1, \ldots, ct_n\} \qquad (Constraints) \qquad (42)$$
$$ct := \langle if fa_1 then MUST fa_2\rangle \mid \langle if fa_1 then MUST \neg fa_2\rangle \qquad (Constraint) \qquad (43)$$
$$fa_1, fa_2 \in Fa$$

An *ASCENS Logical Framework* (Lf)is composed of *facts* (Fa), *rules* (Rl) and *constraints* (Ct) (Definition 37). As shown in Definitions 38 through 43, the Lf's components are built with ontology terms:

- *facts* – define true statements in the ontologies (O);
- *rules* – express knowledge such as: 1) *if H than C*; or 2) *if H than C1 else C2*; where *H* is hypothesis of the rule and *C* is the conclusion;
- *constraints* – used to validate knowledge, i.e., to check its consistency. Can be *positive* or *negative* and express knowledge of the form:
  1) *if A holds, so must B*;
  2) *if A holds B must not.*

Constraints are rather consistency rules helping the knowledge-processing engines check the consistency of a KC (knowledge corpus).

ASCENS KNOWLEDGE BASE OPERATORS

$Op := \{Ask,\ Tell,\ Oop\}$ (*ASCENS Knowledge Base Operators*) (44)

$Ask := retrieve\ (Kc) \longrightarrow Ip \lhd Kc$ (*query knowledge base*) (45)

$Tell := update\ (Kc) \longrightarrow Ip \rhd Kc$ (*update knowledge base*) (46)

$Oop := fo\ (Oi) \longrightarrow Ip \rhd Kc$ (*Inter-ontology Operators*) (47)

$Oi \subset O$

ASCENS INFERENCE PRIMITIVES

$Ip := \{ip_0, ip_1, \ldots, ip_n\}$ (*Inference Primitives*) (48)

$ip := impl\ (FOL)\ |\ impl\ (FOPL)\ |\ impl\ (DL)$ (*Inference Primitive*) (49)

The *ASCENS Knowledge Base Operators* (Op) can be grouped into three groups: Ask operators (retrieve knowledge from a *knowledge corpus*Kc), $Tell$ operators (update a Kc) and *inter-ontology operators* (Oop) intended to work on one or more ontologies (see Definitions 44 through 47). Such operators can be, *merging*, *mapping*, *alignment*, etc. Note that all the *Knowledge Base Operators* (Op) may imply the use of *inference primitives*, i.e., new knowledge might be produced (inferred) and stored in the KB (see Definitions 45 through 47).

The *ASCENS Inference Primitives* (Ip) are intended to specify algorithms for reasoning and knowledge inference. The inference algorithms will be based on reasoning algorithms relying on First Order Logic (FOL) [5] (and its extensions), First Order Probabilistic Logic (FOPL) [15] and on Description Logics (DL) [6]. FOPL increases the power of FOL by allowing us to assert in a natural way "likely" features of objects and concepts via a probability distribution over the possibilities that we envision. Having logics with semantics gives us a notion of deductive entailment. It is our intention to address the following inference techniques inherent in FOL and DL:

- *induction* (FOL) – induct new general knowledge from specific examples;
  Example: *Every robot I know has grippers. → Robots have grippers.*
- *deduction* (FOL) – deduct new specific knowledge from more general one;
  Example: *Robots can move. MarXbot is a robot. →MarXbot can move.*
- *abduction* (FOPL) – conclude new knowledge based on shared attributes.
  Example: *The object was pulled by a robot.*
  MarXbot has a gripper. →MarXbot pulled the object.
- *subsumption* (DL) – the act of subsuming a concept by another concept;
  Example: *Exploit the taxonomy structure of concepts that are defined in the ontology and compute a new taxonomy for a set of concepts or derive matching statement from computed generalization/specialization relationships between task and query.*
- *classification* (DL) – assessing the category a given object belongs to;
- *recognition* (DL) – recognizing an object in the environment.

Note that *uncertainty* is an important issue in abductive reasoning (abduction), which cannot be handled by the traditional FOL, but by FOPL. Abduction is inherently

uncertain and may lead to multiple plausible hypotheses, and to find the right one those hypotheses can be ranked by their plausibility (probability) if the latter can be determined. For example, given rules $A \longrightarrow B$and $C \longrightarrow B$, and fact B, both A and C are plausible hypotheses and the inference mechanism shall pick up the one with higher probability.

5 The Ensemble of Robots Case Study

The ensemble of robots case study targets swarms of intelligent robots with self-awareness capabilities that help the entire swarm acquire the capacity to reason, plan and autonomously act. The case study relies on the marXbot robotics platform [16], which is a modular research robot equipped with a set of devices that help the robot interact with other robots of the swarm or the robotic environment. The environment is defined as an arena where special cuboid-shaped obstacles are present in arbitrary positions and orientations. Moreover, the environment may contain a number of light sources, usually placed behind the goal area, which act as environmental cues used as shared reference frames among all robots.

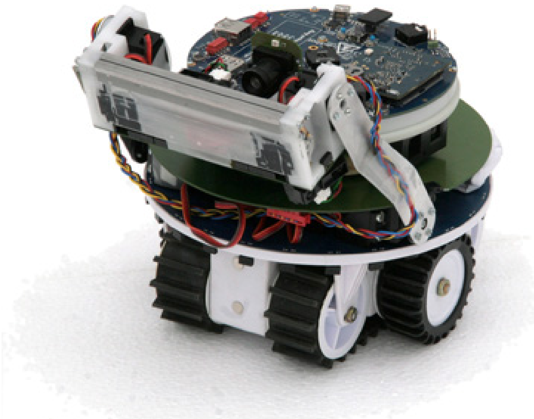


Fig. 2. A marXbot Robot [16]

Figure 2 shows a marXbot robot [16]. Such robot is equipped with a set of devices to interact with the environment and with other robots of the swarm:

- a light sensor, that is able to perceive a noisy light gradient around the robot in the 2D plane;
- a distance scanner that is used to obtain noisy distances and angular values from the robot to other objects in the environment. Its range is 1.5 meters.
- a range and bearing communication system [28], with which a robot can communicate with other robots that are in line of sight.
- a gripper, that is used to physically connect to the transported object;
- two wheels independently controlled to set the speed of the robot.

Currently, the marXbots robots are able to work in teams where they coordinate based on simple interactions on group tasks. For example, a group of marXbots robots may collectively move a relatively heavy object from point ***A*** to point ***B*** by using their grippers.

5.1 The marXbots Robot Ontologies

To tackle the marXbots knowledge representation problem, an initialstructure for the marXbotsRobot Ontology (SC Ontology) has been developed with KnowLang. Figure 3 depicts a concept tree ct(see Definition 8) with a tree root tr“Thing”. The concept “Thing” is determined by the metaconceptCm(see Definition 6)“Robot Thing”, which carries information about the interpretation of the root concept “Thing” such as “*Thing is anything that can be related to a marXbots robot*”. According to this concept tree there are two categories of things in a marXbots robot: *entities* and *virtual entities*, where both are used to organize the vocabulary in the *internal* robot domain. Note that all the explicit concepts Ce (see Definition 13) are presented as concepts in this concept tree – qualified path “Thing→Virtual Entity→Phenomenon”, i.e., in this SC Ontology, the explicit concepts inherit (“↠”) the concepts “Phenomenon”, “Virtual Entity” and “Thing”.

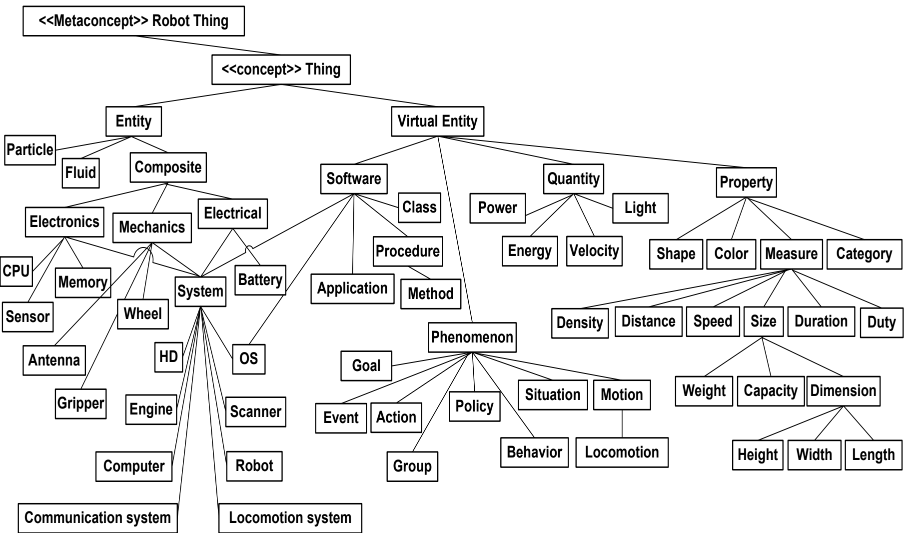


Fig. 3. The marXbots Robot SC Ontology: Robot Concept Tree

Figure 4 depicts an object tree ot (see Definition 27) of the marXbots Robot SC Ontology. As shown, the Robot Object Tree shows the *object properties* of the marXbots Robot object. Note that both the concept and object trees presented here are partial, due to space limitations. Moreover, the marXbots Robot SC Ontology contains a few more concept and object trees, such as the Relations Concept Tree not presented here, etc.

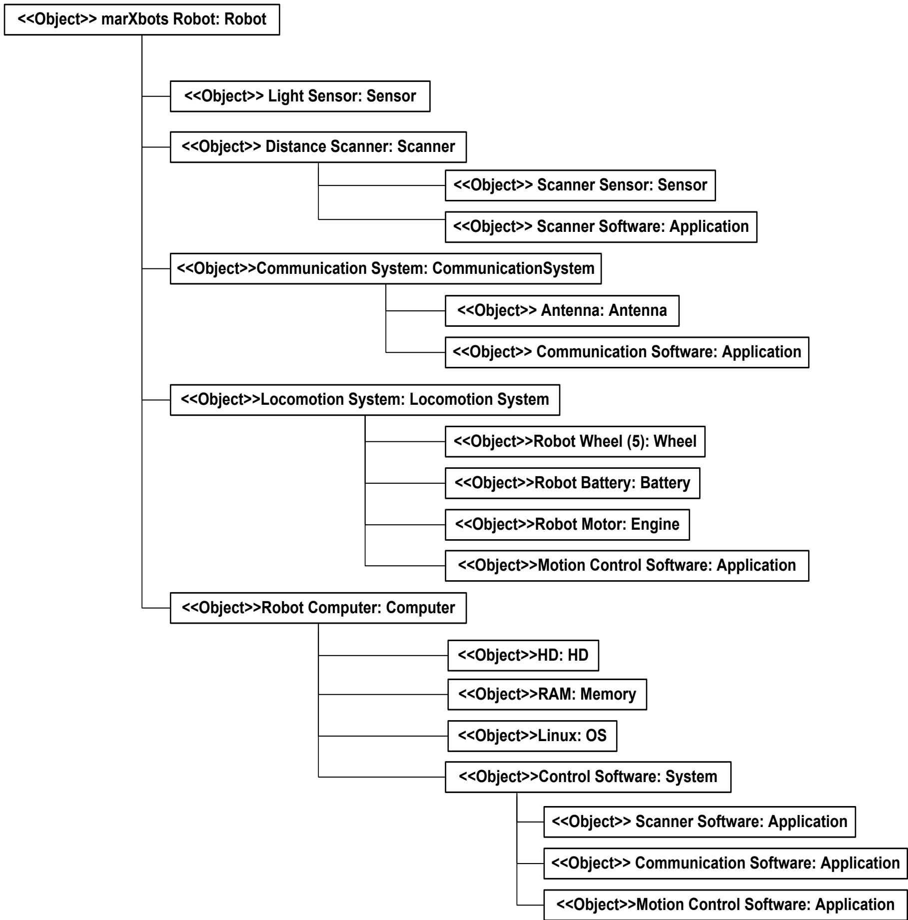


Fig. 4. The marXbots Robot SC Ontology: Robot Object Tree

In addition to the marXbots SC Ontology, to represent the knowledge in all necessary aspects, we have developed initial variants of the other three ASCENS ontologies – SCE Ontology, Environment Ontology, and Situational Ontology (see Section 3). Figure 5 depicts apartial concept tree of the marXbots Robot Environment Ontology. This ontology presents parameters and properties of the robot's operational environment, e.g., external systems (humans, other robots, etc.), concepts (velocity, event, signal, etc.), obstacles, etc. Due to space limitations, the other ontologies with their concept and object trees are not presented here.

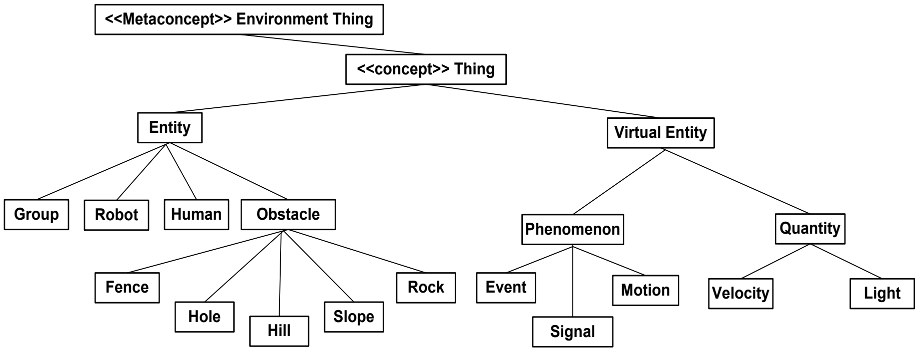


Fig. 5. The marXbots Robot Environment Ontology: Environment Concept Tree

5.2 The marXbots Robot Contexts and Logical Framework

In specific situations, the robot's inferential engine narrows the scope of knowledge in order to reason more efficiently. This ability is supported by the KnowLang's Context constructcx(see Definition 33).

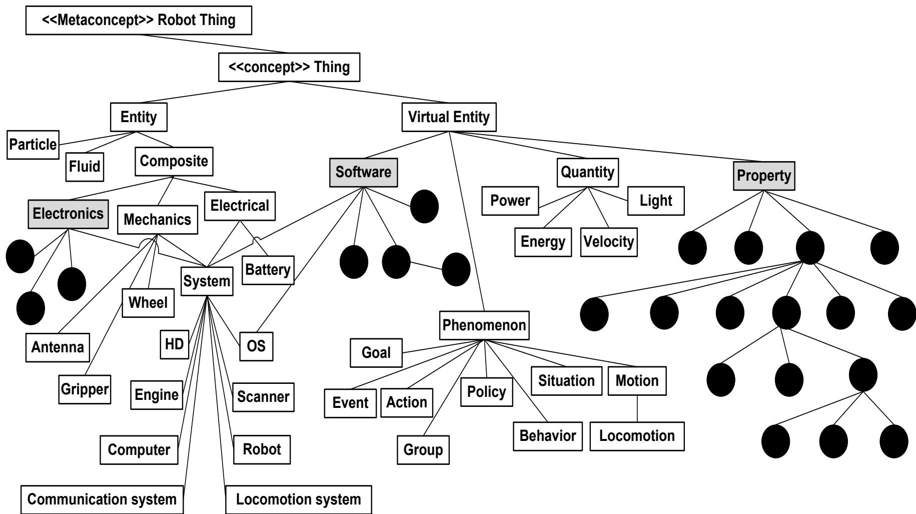


Fig. 6. The marXbots Robot Environment Ontology: Environment Concept Tree

For the purpose of this case study, we have specified a few Contexts, e.g., a context corresponding to the situation "Robot cannot move". This Context $cx$is specified with one single Ambient Tree at as following:

```
CONTEXT name = "Robot cannot move" {
    AMBIENT_TREE {
        ONTOLOGY: Robot;
        CONCEPT_TREE: Thing;
        AMBIENT_CONCEPTS {Electronics, Software, Property}
    }
}
```

This Context is applied automatically at runtime to narrow the scope of knowledge as shown in Figure 6. Note that the ambient concepts define the "depth" of the concept tree in that specific context, i.e., all the concepts descending from those ambient concepts are generalized (or abstracted) by the ambient concepts. For example, we do not deal with Shape, Color, or Measure anymore, but with Property, because the latter is the ambient concept for the former.

When a robot ends up in such a situation, it checks for possible action determined by *policies* (specified by one of the robot's ontologies) (see Definition 15) or by *rules* (specified by the robots' Logical Framework) (see Definition 41). For example, for the purpose of this case study, we have specified two rules as part of the robot's Logical Framework:

```
RULE {
	IF "robot cannot move" THEN {
		DO ACTION "check battery"
	}
}
RULE {
	IF "robot cannot move" AND "battery is charged" THEN {
		DO ACTION "run scanner for obstacles on road"
	}
}
```

6 Conclusion and Future Work

As part of a major international European project, we are currently developing the KnowLang formal language for knowledge representation in a special class of autonomous systems termed as ASCENS. To provide comprehensive and powerful specification formalism, we propose a special multi-tier specification model, allowing for knowledge representation at different depths of knowledge. The KnowLang specification model defines an AKB as composed of a special *knowledge corpus*, *knowledge base operators* and *inference primitives*. The knowledge corpus is built of a *domain ontology*, special knowledge-narrowing *contexts* and a special *logical framework* providing *facts*, *rules* and *constraints*. The *knowledge base operators* allow for knowledge retrieval, update and special inter-ontology operations. All these, may rely on the inference primitives, and therefore new knowledge might be inferred.

Our plans for future work are mainly concerned with further and complete development of KnowLang including a toolset for formal validation. Once implemented, KnowLang will be used to specify the knowledge representation for all the three ASCENS case studies.

Acknowledgment. This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre and the European Union FP7 Integrated Project Autonomic Service-Component Ensembles (ASCENS).

References

1. Makhfi, P.: MAKHFI - Methodic Applied Knowledge to Hyper Fictitious Intelligence (2008), http://www.makhfi.com/
2. ASCENS – Autonomic Service-Component Ensembles (2010), http://www.ascens-ist.eu/
3. Devedzic, V., Radovic, D.: A Framework for Building Intelligent Manufacturing Systems. IEEE Transactions on Systems, Man, and Cybernetics,Part C - Applications and Reviews 29, 422–439 (1999)
4. Vassev, E., Hinchey, M.: Knowledge Representation and Awareness in Autonomic Service-Component Ensembles – State of the Art. In: Proceedings of the14th IEEE International Symposium on Object/Component/ Service-oriented Real-time Distributed Computing Workshops, pp. 110–119. IEEE Computer Society (2011)
5. Brachman, R.J., Levesque, H.J.: Knowledge representation and reasoning. Elsevier, San Francisco (2004)
6. Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.) The Description Logic Handbook (2002)
7. Riazanov, A.: Implementing an Efficient Theorem Prover. Ph.D. Dissertation, University of Manchester (2003)
8. Weidenbach, C.: SPASS:Combining superposition, Sorts and Splitting. In: Handbook of Automated Reasoning. Elsevier (1999)
9. Schulz, S.: E - a brainiac theorem prover. Journal of AI Communications 15(2), 111–126 (2002)
10. RacerPro 2.0, http://www.racer-systems.com
11. Guo, Y., Heflin, J., Pan, Z.: Benchmarking DAML+OIL Repositories. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 613–627. Springer, Heidelberg (2003)
12. European Commission – CORDIS, Seventh Framework Program (FP7), http://cordis.europa.eu/fp7/home_en.html
13. Vassev, E., Hinchey, M., Gaudin, B., Nixon, P.: Requirements and Initial Model for KnowLang – A Language for Knowledge Representation in Autonomic Service-Component Ensembles. In: Proceedings of the Fourth International C* Conference on Computer Science & Software Engineering(C3S2E 2011), pp. 35–42. ACM (2011)
14. Swartout, W., Tate, A.: Ontologies. IEEE Intelligent Systems 14, 18–19 (1999)
15. Halpern, J.Y.: An analysis of first-order logics of probability. Artificial Intelligence 46, 311–350 (1990)
16. Bonani, M., Baaboura, T., Retornaz, P., Vaussard, F., Magnenat, S., Burnier, D., Longchamp, V., Mondada, F.: marXbot, Laborotoire de SystemesRobotiques (LSRO), École Polytechnique Fédérale de Lausanne, http://mobots.epfl.ch/marxbot.html

Object Detection and Classification for Domestic Robots

Markus Vincze, Walter Wohlkinger, Sven Olufs, Peter Einramhof, Robert Schwarz, and Karthik Varadarajan*

Technische Universität Wien, 1040 Vienna, Austria
vincze@acin.tuwien.ac.at

Abstract. A main task for domestic robots is to navigate safely at home, find places and detect objects. We set out to exploit the knowledge available to the robot to constrain the task of understanding the structure of its environment, i.e., ground for safe motion and walls for localisation, to simplify object detection and classification. We start from exploiting the known geometry and kinematics of the robot to obtain ground point disparities. This considerably improves robustness in combination with a histogram approach over patches in the disparity image. We then show that stereo data can be used for localisation and eventually for object detection classification and that this system approach improves object detection and classification rates considerably.

1 Introduction

The purpose of this paper is to propose a systems approach to robotic navigation and object classification in domestic settings to take on these challenges. The inspiration is taken from the holistic approaches [14][12]. The rationale is that the domestic robot accumulates knowledge about the environment when moving safely through it and can exploit this knowledge to enhance its cognitive capabilities. Starting from obtaining the free ground plane in front of the robot, which is necessary for safe navigation, the free space is used for localisation, and the borders of the free space indicate where the typical objects in rooms, such as chairs, start (Figure 1). An advantage is that the detected ground plane gives information about the robot camera orientation, which can be used to obtain better information about locations. The accurate knowledge of the ground also aids more robust object class detection, e.g., for chairs.

One of the open questions in domestic robotics is which sensor(s) to use to obtain 3D information additionally to colour images to cope with the versatile structure present in a home setting. Options are laser range sensors [15] and time of flight sensors [17]. We propose to use stereo cameras and intend to show that the depth and colour information can be used to enable the capabilities summarised in Figure 1. The paper focuses on using stereo images to efficiently

* The research leading to these results has received funding from the European Community for projects robots@home and HOBBIT.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 106–120, 2012.
© Springer-Verlag Berlin Heidelberg 2012

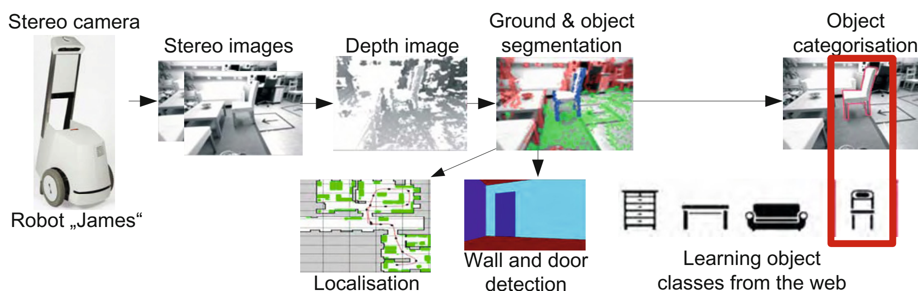


Fig. 1. Diagram of the situated approach to domestic robotics. Ground plane data is exploited for localisation and to constrain subsequent object classification.

and reliably detect the ground plane and on how this information is best used for robust localisation and object classification.

2 System Approach to Domestic Robotics

Figure 1 presented the basic approach. The robot is equipped with two sets of stereo cameras: (1) the top system looks down in front of the robot to determine the drivable free space immediately in front of the robot, and (2) a stereo system about 90 cm above ground (the black box below the white board of the upper body of James in Figure 1) looks forward parallel to the ground to obtain good views from the walls and doors around the robot. Dense depth images are obtained using the approach in [6]. Two sets of cameras are required to obtain a better coverage of the floor as well as the surrounding of the robot. To obtain good localisation the field of view is relatively large with about 100 degrees.

Using the depth image of the top camera the ground plane and the accurate robot roll and pitch angle are detected, Section 3. The ground plane is critical for navigation and to safely avoid all obstacles in a home. But it also places the robot in its environment by clearly indicating where object beyond the border of free space are expected. With this information the detection of objects is considerably simplified. Furthermore, accurate roll and pitch of the robot are used as prior to improve finding of support planes for object categorisation and the detection of rooms structure such as walls and doors.

3 Ground Plane and Object Segmentation

When using a 3D sensor resp. the 2.5D depth images from a stereo camera, the captured scene is represented in the form of a point cloud, where the ground plane connects walls and other objects to one contiguous cluster. Thus, it is necessary to segment scenes into ground and objects before the data can be used for obstacle avoidance and self localisation, or object classification. The first step is to detect and remove the ground plane, which leaves single-standing point clusters representing objects. Furthermore, the dynamic pitch and roll of

the sensor with respect to the world coordinate system can be derived from the plane parameters.

In [2] we presented the details of the algorithm to find ground planes even if the robot tumbles and with ground plane detection correctly adjust its orientation to ground and removes it. Due to space restrictions we only present results of this approach in Fig. 2. All stereo points from one height form a "virtual laser scan" (Fig. 2a and 2c). As stated in the introduction, the 2D representation is compatible to current mobile robotics' obstacle avoidance algorithms.

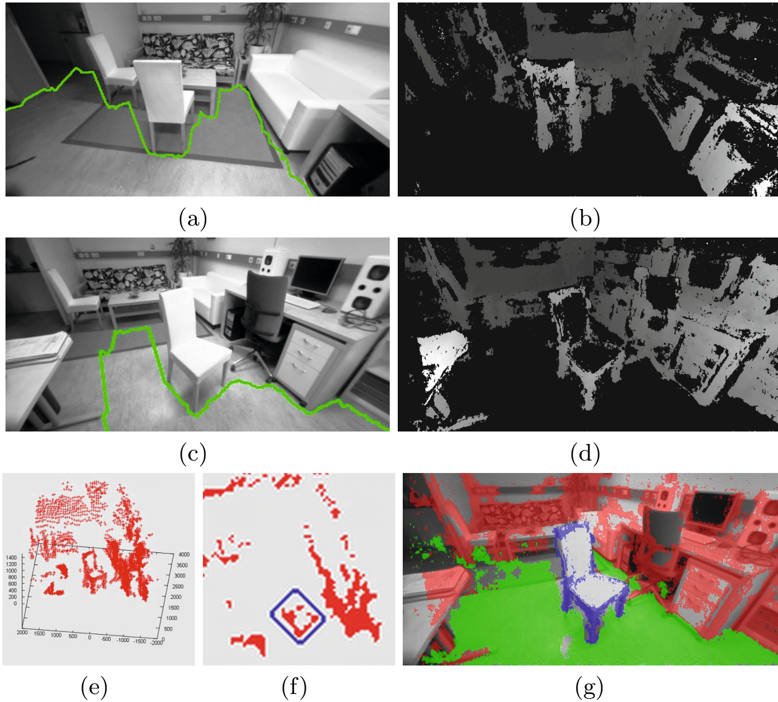


Fig. 2. Ground detection for object classification: (a) and (c) show the virtual laser scan overlayed onto the left rectified camera image. (b) and (d) show the result of removing ground plane pixels from the full resolution disparity image. (e) shows the 3D data calculated from the reduced resolution disparity map's non-ground points. (f) represents an occupancy grid (5cm x 5cm cells) resulting from the projection of the 3D data onto the ground plane; the blue rectangle marks the single-standing point cluster stemming from the chair. Finally, (g) shows the left rectified camera image with the colour-coded disparity map; green marks the ground, red objects (or obstacles) and blue marks the chair (object of interest).

Fig. 2 shows the results for two scenes: In Fig. 2a and 2c the virtual laser scan is overlayed onto the left rectified camera image. Fig. 2b and 2d show the result of removing ground plane pixels from the full resolution disparity image. Fig. 2e-g show the selection of an "object of interest" whose 3D data can be provided to

object classification. For this example we simply took the closest single-standing point cluster in front of the robot. The input disparity maps stemming from a stereo camera mounted in a height of 132cm above the ground and tilted downwards 32 degrees. Computation of the reduced resolution disparity image takes 7ms on average using one core of a notebook with a Core Duo T2250 CPU (1.73GHz).

4 Localisation from Ground Plane

Robust Self-localization is the base for all kinds of behavior-based systems. It has to cope with uncertainty due to imprecise sensors, incomplete knowledge of the environment and limited range of sensing. Very popular approaches for localization are the dynamic state space models using Bayesian Filter variants like MCL [20] or EKF [1]. MCL method has been proven a robust and precise method, but only under certain constraints: The map must contain the complete environment and the sensor data must contain enough "true data" that is known from the map. In fact not all constraints can always be fulfilled due to, e.g., technical restrictions of the sensor or changing environment. A map that does not contain all features as well as limited sensing (Field of View or Range) will lead to unsatisfactory precision.

In this section, we shortly outline the new area-based observation model that tracks the ground area inside a the "free room" (that is, the not occupied cells) of a known map by using a visibility information of a believed pose to the map. In practical experiments carried out with data obtained by a real robot we demonstrate that our model improves the precision and robustness of standard MCL methods when dealing with incomplete maps and limited range of sensing. We use laser and the detected ground plane of the previous section as input in order to demonstrate the robustness of our model. An efficient real-time approximation is given in [11] using *integral images* and image decomposition.

The model is inspired by the work of Thrun [19] on robot mapping using a *ground space* model for local maps. The *ground space model* is used for alignment, but not for localization itself since the ground cannot be transformed into a feature vector like a laser scan. The main idea of our observation model is as follows: We detect the *ground space* area and keep the detected space inside the non-occupied area of the a-priori known map. In the fashion of Bayesian filters and Markov Chains we track and refine the pose over time. It is assumed that the ground can be detected as given in Section 3 rather than detecting a wall. We model the *ground space* in contrast to standard localization approaches in an opportunistic way: We define an observation model that calculates the difference of the observed ground space and the free space according to the map at the believed pose. This approach implicitly deals with unmapped regions as well. As result we obtain possible poses that are not violated (i.e., not wall/unexplored areas are intersected) by the observed area. In combination with a state estimation algorithm and motion model we are able to track the pose. For more details please refer to [11].

To indicate that the usefulness of ground only we compare our proposed area-based observation model with the standard beam model using the well-known Monte Carlo Localization method in practice. In order to demonstrate the high robustness of our model we use also stereo vision as sensor. The beam model with stereo uses an modified Model [3] that also uses wall features. The main difference to the laser beam-model is a that it rejects outliers.

For our experiments we use a non-holonomic mobile robot manufactured by Bluebotics with an additional SICK LMS 200 laser range finder mounted to its front. We use a b/w stereo sensor with vertical camera alignment and approx. 90 degrees field of view. The sensor is oriented into the robot's driving direction and mounted at a height of 30 cm over ground. We use the Videre-Design SRI Stereo Engine for dense stereo-data calculation at a resolution of 640x480 (VGA) and using 96 disparities.

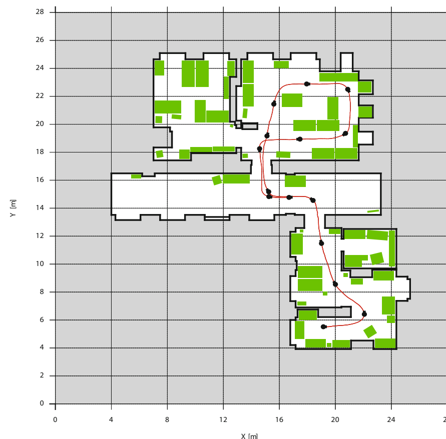


Fig. 3. Sample map of the used environment. The robot is shown as black circle/red line, green areas as furniture and walls as black . The furniture is not mapped.

We choose a typical office environment (see Fig. 3) for camera, odometry and laser data. An external sensor system (multiple vision and laser scanners) for ground truth with a precision of $2cm$ is used. We use two different maps of the same environment: The first map contains the full environment like traditional localization systems [18,3,13]. The second map of the environment consists only of wall and ground segments like the footprint of a building. Please note that the furniture (in green) is only shown for sake of completeness but not included in the map. This is done on purpose to demonstrate the robustness and precision of our approach. We recorded representative eight tours through our lab with a total length of approximately 1600 meters. The robot moves with an average travelling speed of $0.65\frac{m}{s}$. Table 1 shows the average localization error on the complete data set. First we see that the unconstrained laser scanner with a complete map shows a good precision with the Beam-Model and our Model.

Well, this result is not unexpected, MCL has already been proven a robust and precise method under these conditions. We want to emphasise that our proposed model works as good as the usual model under these conditions.

Table 1. Average Localization Error of the complete data

	MCL's Model		Our Model	
Sensor	Translative Error [cm]	Rotative Error [deg]	Translative Error [cm]	Rotative Error [deg]
Laser / full range, complete Map	1.825	1.176	2.135	1.023
Laser / full range, incomplete Map	3.864	4.776	2.253	1.383
Laser / 4m range, incomplete Map	23.396	31.589	1.825	1.041
Laser / FoV 66 deg, incomplete Map	32.975	47.166	1.994	1.175
Laser / Fov 66 deg, 4m, incomplete Map	25.190	21.716	2.736	2.750
Ground Plane full range, incomplete Map	9.427	13.063	4.858	1.993

5 Object Categorisation

To overcome the disadvantages of the environment and target categories, we decided to use a pure-vision based system for the task, namely dense stereo, and develop the system operation from the view of an embodied agent situated in the home environment. We start from the requirement to devise a framework that can be easily extended to new object classes. This has been attempted for appearance of objects in [9] where they showed a rather slow approach used laser scans and domain adaptation. Alternatively, we propose to use only the perfect 3D data and transform it into sensor simulated data to cope with the typical problems of real applications such as only one view of an object (2.5D) with self-occlusion, incomplete models, aliasing effects and realistic noise levels from stereo data. We then use these data to calculate object classifiers extending the 3D Harmonics descriptor [8] with the constraints from the robotics domain and match it against the database to find the nearest class to the object.

Recent results on the Pascal visual object classes challenge 2009[1] show the state of the art in 2D object class recognition. In the detection challenge the results are far below the classification challenge. The best methods gets 47.8% on the aeroplane class and 15.0% on chairs. These results are produced using imagery from flickr[2] as training input.

Most of the systems using real world data make assumptions on the location of the objects, i.e., ground floor detection or detection of other support planes. The work most similar to our work is [10] where a mobile robot equipped with a tilting laser scanner extracts support planes, on which objects are assumed and a subsequent object recognition and classification step is performed on 2D high resolution images taken at these locations. We also have to mention the Sharp

1 http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2009/
2 www.flickr.com

3D SHREC shape retrival contest [3] where the goal is to retrieve similar objects from a database given a range scan. The best method, which uses a SIFT-based approach on a grid spanned over the model, achieves about 50% AP on these high resolution, laser scanned objects. This contest is - to our knowledge - the thematically closest dataset available to test 2.5D object classification on.

5.1 System Approach to Object Classification

We now give an overview of constraints that we exploit as priors in our robotic object classification system.

Scale. Pure image based recognition and classification algorithms do not have access to the scale of objects in the image. Therefore they have to use multi-scale approaches that increase the computational load tremendously. Exploiting a calibrated stereo camera, the scale ambiguity is non-present and algorithmic complexity is reduced such as shown recently in [5].

Orientation. Knowing the sensor-to-robot geometry and assuming regular robot movement regarding to the robot specifications, we can infer the orientation and location of the ground plane to within a certain accuracy.

Room and Object Class. Another priory is to exploit contextual knowledge, for example, about the class of objects to search for. Rooms are likely to contain room-specific objects. As shown in [22] this information can be extracted automatically from the web and proves valuable in object search.

Datasources. A mobile robot equipped with sensors has the option to take images from different views. Additionally, different image characteristics can be exploited: monochrome, color, and 3D data.

5.2 Web-Based Model Aquisition

Data collection for learning new object classes is a time consuming and sometimes tedious work if one tries to get a large number of classes and especially exhausting if the data is not standard imagery but 3D data which has to be created with special hardware. Therefore we propose another approach and tap the huge amount of freely available 3D models from the web.

Web-Download. The input into our model acquisition system is the name of the new object class. With this keyword we search for 3D models on Google Warehouse[4]. Because some object classes have a huge intraclass variability, a large amount of training data is required. Classes such as “dining chair” have a high intraclass variability and need many exemplar models whereas classes such as “apple” only need one or two exemplars to be useful.

[3] `http://www.itl.nist.gov/iad/vug/sharp/contest/2010/RangeScans/`

[4] `http://sketchup.google.com/3dwarehouse/`

Domain Simulation. To use the models from the web, we generate synthetic 2.5D models by rendering and sampling the 3D models from views around the model. We use 45 degree steps in elevation and azimuth which results in 24 views for a model, see Figure 4(left). Figure 4(right) shows the 2.5D partial point clouds created for one model from the chair and mug classes. The most useful information in these figures is the fact that the same object can have a completely different shape and therefore a different representation when seen from different viewpoints.

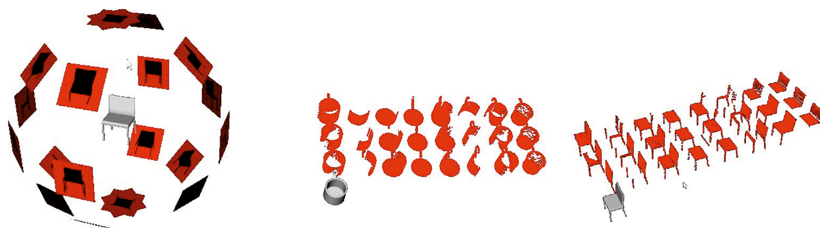


Fig. 4. Model rendering from 24 views at 45 degree steps in elevation and azimuth and the resulting 2.5D point-clouds generated from the web-downloaded 3D model of a mug and a chair.

5.3 Processing Steps

In the following we summarise the processing steps [23][24].

Support Plane Detection. In the first iteration of the data partitioning the ground floor is the basic and elementary support plane. We apply a RANSAC-based plane fitting procedure exploiting the known robot geometry, i.e., the camera pose wrt. the wheels, and we use the two points where the wheels touch the ground as two of the three points in the plane candidate search. This gives more robust results in cases where the ground plane is not the dominant plane in the scene. Figure 5 shows the initial 3D point cloud with the floor detected and coloured in red.

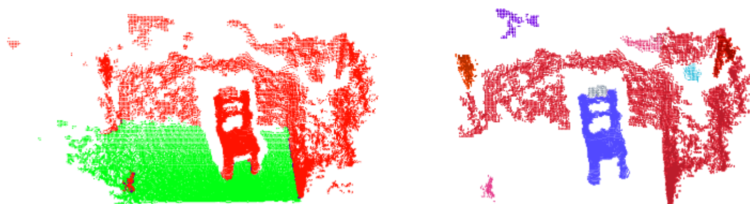


Fig. 5. Floor detection result (left) and point cloud clusters

Clustering. The next step is to obtain groups of data points that may indicate individual objects. To this end we use clustering based on flood filling on the remaining points above the ground plane. The size of the filter kernel depends on the size of the smallest object we are looking for and the minimal allowed

distance between two objects. As we already have the object classes we can extract these parameters from the database. For the experiments, the kernel diameter is set to 5cm. Results of this procedure can be seen in Figure 5 (right).

Region Prior Filter. One prior is the assumption that objects are located only on support planes. Hence, an efficient way of reducing the candidate clusters for object classification is to reject all clusters outside the support region. To obtain the boundary for the support plane, the 2D convex hull is calculated and the projected point clusters are tested against the plane polygon using the odd-even test. The support plane boundary is shown in red in Figure 6 (left) with the projected clusters in green. Only clusters inside this support plane are valid candidates for the next processing step.

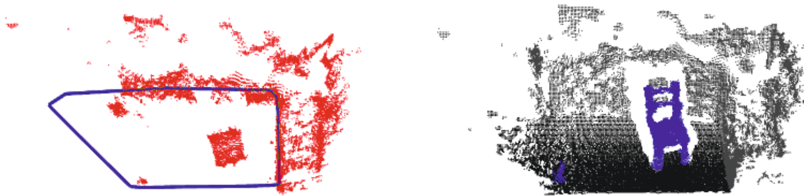


Fig. 6. Support plane boundary and candidate objects for classification

Supported Candidates Filter. The "object on support plane" assumption also implies that the candidate objects are actually "on" the support plane. We therefore only process point clusters which are attached to their support plane. The resulting object candidates for the classification produced by the data acquisition chain are shown in red in Figure 6

Play It Again, Sam. Up to this point, the only support plane extracted from the data is the ground floor. For finding objects on further support planes – such as objects on tables and counters – the RANSAC-based support plane detection and the associated clustering, region prior and candidate filtering stages are performed again on the clusters.

5.4 Classification

The goal of classification is to find the correct class label for a given data cluster. This can also be seen as finding the most similar object to the query data and assigning the label of the most similar match. Finding similar objects - especially 3D models in large databases - with efficient and robust algorithms has attracted a good amount of researchers over the last years. Following the proposed robotics approach to object classification, we want to stress the following properties and differences to classical approaches [4][7].

3D Descriptor Properties. The biggest challenge in 3D shape matching is the fact that objects should be considered to be the same if they differ by a similarity transformation. To explicitly search over the whole space of transformations is

impracticable because efficiency is a key property a retrieval algorithm should have. Hence, the similarity metric must implicitly provide the similarity at the optimal alignment of the two models.

Normalization. This can be achieved in a normalization step prior to the similarity calculation for each model where translation, rotation and scale are normalized, i.e., a canonical frame is computed.

Invariance. To avoid the imperfections of a prior normalization step, the alternative is to design the descriptor in a transformation invariant fashion. This means that the descriptors produce the best similarity measure for any transformation. Because of all the difficulties with normalization, we propose to apply an invariant descriptor.

Spherical Harmonics Descriptor. The spherical harmonics descriptor[8] is a affine invariant descriptor which is calculated from a 64x64x64 voxel grid. The calculation of the descriptor from a coarse voxel grid gives us the needed generalization for object class recognition. The descriptor is represented and stored as a 2D histogram in the database which enables us to efficiently calculate the k-nearest neighbours using the Euclidean distance between the query and all entries in the database.

5.5 Results

Tests are shown on datasets collected in our lab on chairs and on a table scene. The chair classes in our system are “dining chair”, “office chair” and “arm chair”. For chairs, the system is able to produce classification results useful for robotic purposes. Our test-database consists of 119 point clouds where 47 chairs are present. The segmentation stage was able to successfully extract 30 of the 47 chairs from the scene. A representative sample of test scenes with chairs in arbitrary position and distractor objects can be seen in Figure 8. Figure 7 shows the precision recall curve for the dining chair class and Figure 7 when testing against the meta class of chairs, consisting of dining-, office- and armchairs. The decrease in performance is only marginal for chairs as to be seen in Figure 7 as they provide enough structural shape to differ from other object classes. This is no longer true for primitive shaped objects like bottles or mugs as the only distinction between a round paper basket and a mug - when seen from a certain viewpoint - is their size, despite their common location. This can be seen in Figure 9 where there is just not enough data for distinguishing between some small objects. Nevertheless most of the mugs on the table were correctly identified.

The presented system heavily depends on good and complete data. Without sufficient data the support planes can not be extracted and the clustering of the object will produce multiple fragments which renders the classification with global desciptors useless.

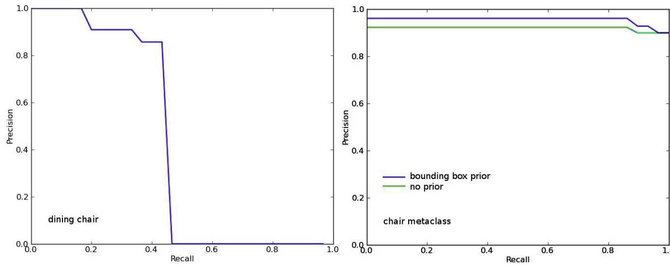


Fig. 7. Left:Precision recall curve for the dining chair class. Right:Precision recall curve for the meta class chair. The blue curve shows the performance of the spherical harmonics descriptor with a bounding box prior, green without.

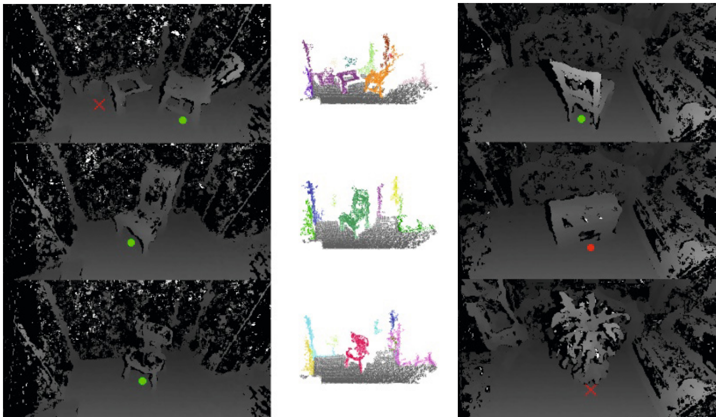


Fig. 8. Test scenes with chairs and distractor objects. Green dots indicate correct class, red dot indicate false positive and red cross indicates not detected as belonging to class chair. Image 2:Chair lying on the ground could not be classified due to insufficient segmentation; Image 6 the box was wrongly classified as belonging to the class chair.

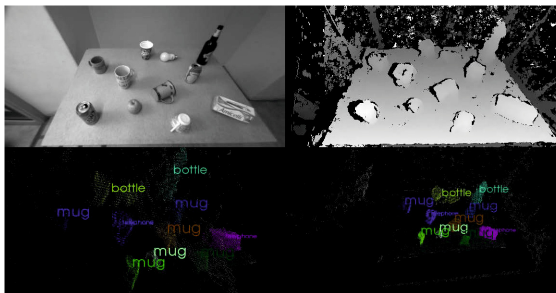


Fig. 9. A tablescene with instances of classes mugs, cans, bottles, apple and light bulb

6 Wall and Door Detection

The previous Section presented an elegant approach to learn object classes from models found on the web. However, there are domestic object classes that are not suited to this approach such as doors and walls. These vertical structures omnipresent in rooms are better obtained from tailored approaches. In an attempt to obtain robust detection results, we propose a twofold approach. Based on colour and dense stereo features we can obtain the wall layout including door openings (Section 6.1).And in a complimentary we investigate the use of vertical line features in a sparse stereo approach (Section 6.2).

6.1 Wall Detection, Room Modeling and Doorway Detection from Colour and Stereo Data

Detection and classification of structural components in the indoor scene assumes significant importance, especially with respect to robot navigation, place learning and semantic object recognition. Due to space limits, we present results and refer the reader to [21] for more details. The idea is based on exploiting constraints for indoor wall, room and door classification. Walls are typically characterized by homogeneous regions or areas with regular texture, usually with high numeric intensity values. The largest single color regions in a given scene, especially with no large occluding obstacles in the vicinity, and walls hold pixels with the farthest visible range information on planes parallel to the ground plane. Rooms are characterized by a combination of walls approximating a cuboid and the largest and most consistent of all possible cuboids in the scene (helps exclude walls internal to the room). Room fitting can be reduced in most cases (based on assumptions of known floor and ceiling) to fitting of a maximum of just three (largest) vertical walls. Finally, doorways are characterized by external outliers (or exclave points in range images) to the room model that can be grouped to form regions with size bounds similar to that of typical doorways.

In [21] we have demonstrated the benefits of a novel framework of fusing 2D local and global features such as edges, textures and regions with geometry information obtained from pixel-wise dense stereo for reliable 3D indoor structural scene representation. The strength of the approach is derived from the novel depth diffusion and segmentation algorithms that result in better surface characterization as opposed to traditional feature based stereo or RANSAC plane fitting approaches. It should also be noted that in the context of indoor 3D room reconstruction, the presented framework is (a) highly efficient with extremely sparse range data (b) preserves and detects depth edges in regions where there are no visible edges in the color data and (c) handles shadows and specular highlights effectively. Examples of using the assumptions in the shortly outlined approach are given in Fig. 10.

6.2 Feature-Based Door Detection

Door detection is again based on the robots knowledge about its vertical pose. The approach consequently searches for vertical line of certain height and

Fig. 10. Results from test environment (Top to bottom) (a) Input scenes (b) 3D model reprojected on to the image plane (c) Ground truth. The percentage of mislabeled pixels were 5, 4, 17, 12 and 5 respectively.

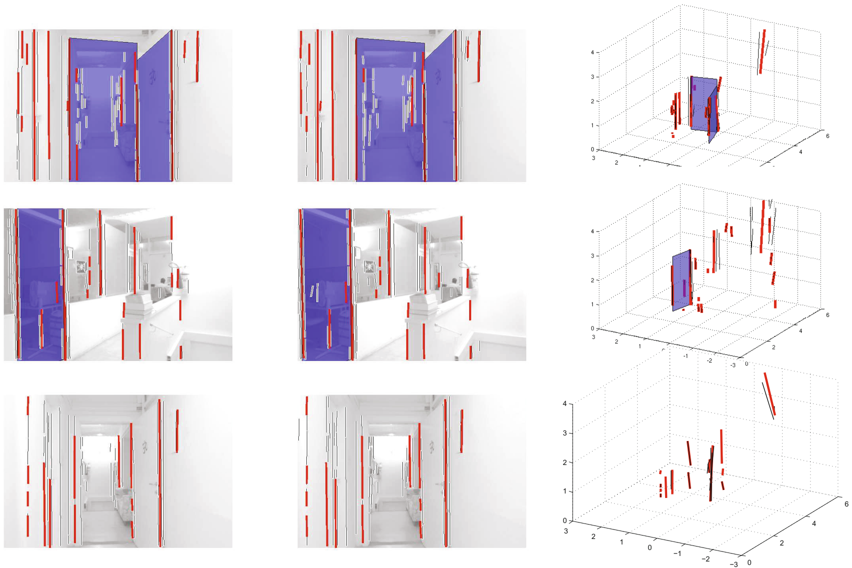


Fig. 11. Results for door detection: first two rows show successful detection, last row shows false line matching with false grouping

parallelism. The main problem is to cope with fragmented lines. The main idea is to post process the 3D lines obtained from stereo processing and limited to vertical lines in Euclidean 3D space based on the mean shift algorithm. Due to space limits we refer the reader to [16] and present a result for completing object detection in domestic settings in Fig. 11.

The grouping algorithm has only the kernel as parameter, so after generating the kernel which size depend on the distance, no other parameter must be set.

Doors are detected where two vertical parallel lines have a distance between 0.6 m and 1.4 m, and both lines have a height of more than 1.5 m. The data set consists of a test run through our office, where several doors can be found. Figure 11 shows example frames: in the first three rows door hypothesis are found and drawn as blue rectangles. The last row shows false grouping and false matching of lines, so the segmented line can not be mapped and grouped in 3D and no door can be found.

7 Conclusion

The aim was to show how the combination of contextual knowledge helps to build a robot suitable for deployment in domestic environments. The idea of this approach is to overcome the difficulties encountered in approaches that do not take into account knowledge available from the agent's environment. The core functionality a home robot needs to provide is to cope with the diversity of furniture in homes. While it is unclear if scanning laser sensors, time-of-flight, triangulation sensors or stereo imaging will prevail in the long run, we propose a complete approach based on stereo sensing. With appropriate adaptation to the specific sensor characteristics the algorithms will work also for other type of depth sensors.

Starting from the basic requirement to obtain the free space in front of the robot for safe navigation and obstacle avoidance, we developed a method to segment the ground plane around the robot (Section 3). In Section 4 we then show how the free space can be exploited to localise the robot. The most important finding is that the free space model compensates disadvantages of the smaller viewing angle as compared to laser range sensors. On the other hand, stereo processing allowed us to take objects at all heights into account. The ground plane is used to boost object classification to over 90 percent for chairs as compared to 15 percent in VOC. And finally we indicate that also wall, door and room structure detection profit from this systems approach.

References

1. Arras, K., Castellanos, J., Schilt, M., Siegwart, R.: Feature-based multi-hypothesis localization and tracking using geometric constraints. Robotics and Autonomous Systems 1(44), 41–53 (2003)
2. Einramhof, P., Vincze, M.: Stereo-based real-time scene segmentation for a home robot. In: International Symposium ELMAR (2010)
3. Elinas, P., Little, J.: omcl: Monte-carlo localization for mobile robots with stereo vision. In: Proceedings of Robotics: Science and Systems, Cambridge, MA, USA, pp. 373–380 (2005)
4. Golovinskiy, A., Kim, V.G., Funkhouser, T.: Shape-based recognition of 3d point clouds in urban environments. In: ICCV (2009)
5. Helmer, S., Lowe, D.: Using stereo for object recognition. In: ICRA (2010)
6. Humenberger, C., Zinner, C., Weber, M., Kubinger, W., Vincze, M.: A fast stereo matching algorithm suitable for embedded real-time systems. Computer Vision and Image Understanding 114, 1180–1202 (2010)

7. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Transactions on Pattern Analysis and Machine Intelligences 21(5), 433–449 (1999)
8. Kazhdan, M., Funkhouser, T., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation of 3d shape descriptors. In: SGP, pp. 156–164 (2003)
9. Lai, K., Fox, D.: Object detection in 3d point clouds using web data and domain adaptation. International Journal of Robotics Research (2010)
10. Meger, D., Gupta, A., Little, J.: Viewpoint detection models for sequential embodied object category recognition. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 5055–5061 (2010), doi:10.1109/ROBOT.2010.5509703
11. Olufs, S., Vincze, M.: An efficient area-based observation model for monte-carlo robot localization. In: International Conference on Intelligent Robots and Systems IROS 2009, St. Louis, USA (2009)
12. Pfeifer, R., Lungarella, M., Iida, F.: Self-organization, embodiment, and biologically inspired robotics. Science 318, 1088–1093 (2007)
13. Plagemann, C., Kersting, K., Pfaff, P., Burgard, W.: Gaussian beam processes: A nonparametric bayesian measurement model for range finders. In: Robotics: Science and Systems (RSS), Atlanta, Georgia, USA (2007)
14. Pylyshyn, Z.: Visual indexes, preconceptual objects, and situated vision. Cognition 80, 127–158 (2001)
15. Rusu, R.B., Blodow, N., Marton, Z.C., Beetz, M.: Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2009)
16. Schwarz, R., Olufs, S., Vincze, M.: Merging line segments in 3d using mean shift algorithm in man-made environment. Austrian Association for Pattern Recognition (2010)
17. Swadzba, A., Wachsmuth, S.: Indoor scene classification using combined 3d and gist features. In: Asian Conference on Computer Vision, Queenstown, New Zealand, vol. 2, pp. 725–739 (2010)
18. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics, 1st edn. MIT Press, Cambridge (2005)
19. Thrun, S., Fox, D., Burgard, W.: A real-time algorithm for mobile robot mapping with application to multi robot and 3d mapping. In: International Conference on Robotics & Automation, San Francisco, CA, USA (2000)
20. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust monte carlo localization for mobile robots. Artificial Intelligence 128(1-2), 99–141 (2000)
21. Varadarajan, K., Vincze, M.: 3d room modeling and doorway detection from indoor stereo imagery using feature guided piecewise depth diffusion. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2010)
22. Viswanathan, P., Meger, D., Southey, T., Little, J.J., Mackworth, A.: Automated spatial-semantic modeling with applications to place labeling and informed search. In: CRV (2009)
23. Wohlkinger, W., Vincze, M.: 3d object classification for mobile robots in home-environments using web-data. In: IEEE International Workshop on Robotics in Alpe-Adria-Danube Region RAAD (2010)
24. Wohlkinger, W., Vincze, M.: Shape-based depth image to 3d model matching and classification with inter-view similarity. Submitted to IEEE IROS (2011)

A Software Integration Framework for Cognitive Systems

Michael Zillich, Wolfgang Ponweiser, and Markus Vincze

Automation and Control Institute, Vienna University of Technology
{zillich,ponweiser,vincze}@acin.tuwien.ac.at

Abstract. Handling complex tasks with increasing levels of autonomy requires robotic systems to incorporate a large number of different functionalities at various levels of abstraction, such as localisation, navigation, object detection and tracking, human robot interaction including speech and gesture recognition as well as high level reasoning and planning. The interaction between functionalities in these cognitive robotics systems not only requires integration at a technical level but more importantly at an organisational and semantic level. Within this contribution, these cognitive functionalities are encapsulated in software components with the objective to provide clearly specified interfaces to allow reuse in other cognitive vision or robotics systems. To arrive at the level of building a system from these functionalities, it is considered essential to provide a framework that coordinates the components. Two principles organise the components: (1) the service principle uses a "yellow pages" directory to announce its capabilities and to select other components, and (2) the hierarchy principle orders components along data abstraction from signal to symbolic levels and ascertains that system response is reactive. The proposed system is demonstrated in a context-oriented system for activity interpretation involving functionalities such as tracking, object and gesture recognition, spatio-temporal object relationships and reasoning to extract symbolic activity descriptions.

1 Introduction

Computer vision has reached a state where it becomes possible to derive conceptual information from basic visual input streams. Applications in the area of mobile robots, surveillance or vehicle steering are some examples. This advance led to the use of terms such as Cognitive Vision, Cognitive Vision System (CVS) and, finally, Cognitive Systems, where emphasis on vision as the sole or main sensor is removed. Such "systems", and in particular, Cognitive *Vision* Systems become possible with the advance of computing power, the availability of techniques to handle the vast amount of video data, the development of reasoning techniques to approach the semantic level, and finally the advances in robotics and other forms of active vision and active embodiment. As a consequence, a CVS or a Cognitive System comprises processes ranging from low-level (data) to high-level (semantic) processing including a large number of models

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 121–135, 2012.
© Springer-Verlag Berlin Heidelberg 2012

and techniques (e.g., perception-action mapping, recognition and categorisation, prediction, reaction and symbolic interpretation) and feedback and interaction mechanisms and embodiment to fulfil a task (e.g., navigating to goals, finding objects and communication to humans).

The ultimate driving force of developing such systems is to obtain systems that are applicable to a wide range of tasks, robust to changing and possibly new environments and situations, and which execute the tasks within reasonable time limits. These expectations in a CVS impose a number of constraints:

- The system is *on-line* with the observed scene. Therefore the system has to *react* in the temporal order of activities observed and with appropriate time delay (*data-driven*).
- The system has to solve several tasks *concurrently*, tasks must be executed on *restricted resources*, tasks are *complex* in terms of the methods and combinations applied and require *control and integration* of (nowadays) *probabilistic* methods to fulfil the demands of cognitive processes. Therefore the system has to enable *task-driven* processes.
- Additionally all practical aspects of software engineering have to be taken into account such as *scalability*, *ease of use* and *software reuse*.[1]

To comply with all these requirements vision scientists have been investigating different architectures and frameworks. Early examples are the VAP project (Vision as Process, [5]), which basically enabled data-driven processes and active vision, or the Khoros software development environment [10]. Later examples are DACS [7] or RAVL [6]. Recent attempts have been made in the robotics, multi-agent and software engineering areas, each area with different approaches and fulfilling a slightly different spectrum of requirements (for details see Section 2).

This work wants to foster the use of a software framework as a tool to build Cognitive Vision *Systems*. This framework has been devised specifically for CVS to study the interaction between low-level and high-level processes. It exploits experience of software engineering to build complex systems based on the component principle [20]. The idea is to develop components individually and to endow the component with a service description that enables other components to *reuse* individual functionalities (or cognitive vision skills) already developed and to focus on studying the interaction in the *system*. Two aspects greatly support a component-based approach. First, experience in software engineering found this as the best solution to reuse software and to build larger systems. And second, it will be shown below that the component structure forces scientists and engineers to clearly specify what their vision function can do under certain circumstances. This description will use the notion of Quality of Service (QoS), which, for the first time, makes computer vision developers specify in formal terms what their algorithm really can do. We see this need to be objective about vision capabilities as a big step forward to advance the science of computer vision and cognitive vision.

[1] The latter two have been often underestimated but they are critical for successful application and everyday use.

After reviewing in more detail related work (Section 2) the software framework is introduced in detail in Section 3. It has been designed for CVS applications, in particular, the ActIPret project, which sets out to interpret activities of persons handling tools and involves skills such as tracking, object and gesture recognition, spatial-temporal object relationships and reasoning to extract the symbolic description. We will exemplify the use of the framework with two examples from this project (Section 4) and finalise with a discussion of the usefulness of such a framework (Section 5).

2 Related Work

To comply with all these requirements it is proposed to use a software framework that provides the basic functionalities for these requirements. One such attempt has been made in the European IST Project ActIPret, which has the goal to interpret activities of a person handling objects. To study the interaction between low-level and high-level processes, a software framework has been developed based on the component principle [20]. The coordination requirements of component based programming are related to distributed artificial intelligence and multi-agent theory [4], [23], [17], [9], [13]. Multi-agent systems use negotiation to determine a configuration. Such an approach is too time consuming for a reactive system that must respond with a fixed delay.

Some architectures for robotics and perception cope with this constraint and are related to component-based system programming (e.g., [1], [2], [11]). Such architectures provide mechanisms to configure a system set-up during compile or start-up time. Dynamic component selection and activation is not permitted in such systems. E.g., the OROCOS project (www.orocos.org) provides basic common principles for frameworks for developing robot control software (e.g., fusion and reactive behaviour). Other tools to develop frameworks can be separated into the communication middle-ware, e.g., RPC (Sun Microsystems), DCOM or CORBA, and higher-level framework tools and concepts such as SAPHIRA (SRI), AYLLU (ActivMedia), BERRA [11], OSCAR [3], SMARTSOFT [14]. However, the API's (application programming interface) of these tools provide little or no temporal dynamics in the sense of on-line changes in the component set-up and trivial or specific API's for sensory processes. The well known robot operating system (ROS) [15] offers a large repository of drivers and components as well as a strong research community. It is intended however to provide only a technical integration framework rather than propose an organisational principle. The CoSy Architecture Schema Toolkit (CAST) [16] in contrast offers a software framework for a variety of languages (C++, Java, Python) built around a distributed blackboard architecture. Components are organised in sub-architectures each with its own working memory, serving as the central knowledge repository for this part of the overall architecture. Keeping data consistent with several components accessing asynchronously can however become problematic.

The ActIPret framework sets out to resolve these shortcomings. It is presented in Section 3 and examples are given in Section 4.

3 A Software Framework for Cognitive Vision Systems

One of the goals of ActIPret is to develop a general-purpose framework for CVS that can be used not only for the ActIPret demonstrator but also in other CVS applications, i.e., other machine vision systems and also in robotic systems. This section summarises the important design goals that are derived from the essential elements of cognitive vision demands as reviewed in Section 1 (Section 3.1). We then give the design decisions to realise the framework (Section 3.2), outline the two particular principles the framework builds on, namely the 'service principle' and the 'hierarchy principle' (Section 3.3), and introduce the structure of the individual components (Section 3.4) and the mechanism to select components (Section 3.5). Finally, Section 3.7 presents important implementation details.

3.1 Requirements on a Software Framework for CVS

The realisation of the CVS goals listed in the Introduction and of systems reviewed impose a series of specific requirements on a software framework. First, computer (and cognitive) vision implies image data of multiple redundancies: temporal redundancy from data streams, stereo or multiple views, within each image many cues and a vast number of features and an even larger number of feature operators, and a multiple of representations to store and formalise the image content. And second, the vision functionalities are not alone. There are many other sensors, e.g., speech, sonar, laser rangers, to name only a few, that contribute the the systems. The system is embodied, in the minimum with the restrictions to camera views. It is embedded in a task and the situations deriving from this, and it may contain knowledge representations or even a kind of common sense knowledge to better cope with these situations. Finally, a CVS is expected to contain a linguistic components that can present knowledge semantically.

To summarise, a software framework for CVS should support to build a system with the following specifications.

- Proactivity: an on-line software system, such as a CVS and in particular the ActIPret system, has to deal with limited resources. Hence the system has to focus its resources (processing power, views, etc.) according to task relevancy. Proactivity subsumes goal and task orientation, taking the initiative to control resources and to generate appropriate sub-tasks [23].
- Reactivity: the system has to react to the sensor input provided by the cameras. The goal is to build a CVS that reacts and responds to human actions. Hence reactions must take place in a time scale appropriate to human activity.
- Scalability: this should be linear or as close as possible to linear. In practice this means that duplication of any component should result in (as near as possible) a duplication of the associated service assuming the existence of similarly duplicated resources.

– Control: scalability also implies distributed control throughout the system (a single control component would not scale). Hence, each component has its own "control policy" (albeit at different scales of task relevance) to control the services requested and the results or data obtained.
– Modularity: the development effort for CV software can be limited by reusing software segments. Modularity forms the basis for software reuse and a dynamic system structure.
– Independence of components: a component only needs to know about the functionality (the service) of another component and not about their implementations. Hence, every component has the control of the services it provides to other components and the services it requests from other components.

Several of these requirements aim at fulfilling a point that is often neglected: (re)usability by the researcher and engineer. Project success depends to a larger extend than generally acknowledged on a fast learning curve, the support to evaluate components and the whole system, transparent use of the tool (e.g., interface specifications), and a certain flexibility to enable modifications without the need to adapt the complete system. Experience from software engineering advocates programming based on *components* the present most successful approach in this respect [20].

3.2 Design Decisions

To realise the framework, some design decisions had to be taken. The decisions are driven by the list of requirements given previously.

– The system is distributed: vision and interpretation require significant processing power. In order to achieve scalability the system has to be distributed over several machines. Distribution into distinct modules (components) simplifies parallel development.
– The components of the framework run asynchronously: the effort required to synchronise a distributed system outweighs the simplification in the integration process. Furthermore, synchronisation is pointless for components having different temporal behaviours such as tracking and recognition.
– Data consistency: control over service responses from other components assumes that the data received is consistent. For example, if two detectors on different cameras are asked to find a hand, the responses must be fused in the component that requested those two services.

3.3 Design Principles

As a consequence of the requirements and following the design decisions, the framework is built according to two major design principles: (1) the "service principle" and (2) the "hierarchy principle". These two terms are explained subsequently. The underlying software concept is founded on a component-based

approach [19], which is faster than classical agent systems to guarantee reactivity and enables easier distribution than conventional object oriented approaches whose scalability ends at the one-computer border.

Our first goal for the Cognitive Vision (CV) framework is to build a task driven system, so every distributed component is task driven. From the point of view of a component, it has to provide one or more interfaces to make its functionality accessible. Using this interface other components can initiate one or more tasks. Components can be regarded as black boxes and so according to the goal of component independence these interfaces have to describe all of the component's capabilities. Because of the task related nature these interfaces are called *services*, which are derived from the CORBAservices ([12]). They form a "Yellow Pages" dynamic look-up directory that makes it possible to select on-line the best service available according to a formalised performance characterisation. The component itself builds the frame for the service provider.

Any component or service provider can use the services of other components. In this case the component or service provider establishes the interface of the service requested. This interface object is called service requester. Every component 'presents' its own abilities by providing services via the service provider and every component makes use of the abilities of other components by requesting services via the service requester. Using the service principle the implementation is hidden outside of the component.

The "hierarchy principle" resolves the problem of building a reactive system. Fully agent-based systems do not use a hierarchy. All agents negotiate until a configuration is reached. This is thought too time consuming for a reactive system. To reduce administrative communication overheads during decision-making (service selection) a strict hierarchical structure of components is used. Hence every link between components has a higher-level component and a lower-level component. The higher-level component always orders the task (service), the lower-level component has to process (provide/deliver) it.

3.4 Component Structure

At this point it is appropriate to define a component [21].

- The inside of a SW-component is a piece of software with some properties. It is a unit or entity that can be reused.
- The outside of a SW-component is an interface with some properties. It provides a good or service to humans or other SW-components.

From the view point of a CVS, a component encapsulates (that is the relation between inside and outside) the cognitive vision (or other system) function, and provides the means for communication with other components. The function itself encompasses all aspects typical of today's vision functions. This includes memory up to the point of a separate component (such as a model server), mechanisms for self-evaluation (e.g., reporting confidence measures, expected accuracy, and resource demands), the necessary control to execute the function,

which entails control of processing as well as views and embodiment and the use of other components, and finally techniques to exploit and report context.

The context states how the software is to be managed and used, within a defined process for software development and maintenance. If the context is not stated (and it usually isn't), then concepts such as encapsulation and reuse are ambiguous [21].

The last point leads to a side effect of using a framework to build a CVS: the contextual relationships, that is how to use the component and what it can do under the contextual circumstances, need to be formalised. While it might be considered cumbersome, this enforces to clearly specify what a vision function can do. And it can be said, with justified self criticism, that scientific work in computer vision has not addressed this issue sufficiently (with the noticeable exception of work in performance evaluation).

In conclusion, using the two design principles (3.3) a component is made up of three elements (see Fig. 1):

- The component itself that is the frame for all other building units. The component unit is responsible for managing services. This includes offering of services that can be provided, reception of service requests and establishing of service providers. Also long term memory of common data or other functionality all providers share is placed in the component.
- The service provider contains the real functionality that provides the service. The component establishes the service provider as an instantiation of a service. consistence.
- The service requester is the access point for all responses from a service provider. It can be established by a component or a service provider that requires other services.

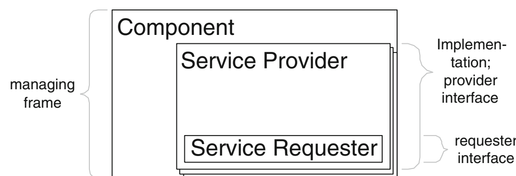


Fig. 1. Structure of a component

3.5 Service Selection

In order to fulfil task orientation and to optimise resource allocation it is necessary to manage service requests. Otherwise components might be activated that cannot obtain the necessary resources. The critical issue is the method of selecting a service. Service selection serves the purpose of selecting resources efficiently. [8] presents an overview of options of making the preferences of the requester and the properties of the provider known to the other components

and/or a middle agent. Often used approaches are blackboard systems, where the preferences are known by all others, or a broker system, where the middle agent knows about both preferences and properties and negotiates to select a service.

The mechanism introduced here is derived from the CORBA trader service [12]. It adheres to the yellow-pages principle, where the provider makes his properties known to the middle agent and the requester. The following sequence describes the process for registering a service in the service list and selecting the service(s). Also refer to Fig. 2 for an example sequence to set-up a component connection.

1. During the component start-up every component registers its services in a global service list, (i.e., they export a service offer). A service list entry consists of the service name, the ID of the component providing the service and an abstract description of the service (i.e., Quality of Service, costs, constraints, etc.). If due to changes in the environment properties of the component change, this is reflected in an updated service list entry.
2. If a component or service provider needs a service of another component it creates a service requester to establish a connection. All selection, configuration and communication is done by the service requester.
3. The service requester starts the search for a service in the service list (it asks for service "S1" in Figure 2). The component description (QoS, constraints, etc.) makes it possible to select the relevant subset of all components providing the service "S1". The service list returns the list of corresponding service offers (in our example, for components 2, 3 and 4). The service requester can now use the service properties to select the service that is most suited to the demand, in this example the service of Component 3.
4. The service requester selects one of the service offers received and establishes a link to the provider selected. This completes the process of service selection.

3.6 Quality of Service, Utility and Costs

The goal of a CVS is to fulfil a task by using appropriate components. The appropriateness can be measured as the Utility (U) of a service. The Cost (C) of running a service is the Utility of the disabled service(s). This stems from the fact that for a given computing power services need to be negotiated according to the utility they offer versus cancelling other services (and their utilities).

The obvious system goal is to optimise system utility, that is, select the service that has the property

$$\max(U - C). \tag{1}$$

Using the notion of QoS, the utility of one component is a function of QoS. The Quality of Service (QoS) defines the properties of components in terms of (1) execution time (as expected or inferred from previous use), (2) accuracy (spatial, 2D or 3D) and (3) the confidence or certainty in a correct match or correct result of the function (also see [18]). As a start the simple relation

$$U = f(QoS) = f(\text{execution time}) + f(\text{accuracy}) + f(\text{confidence}) \tag{2}$$

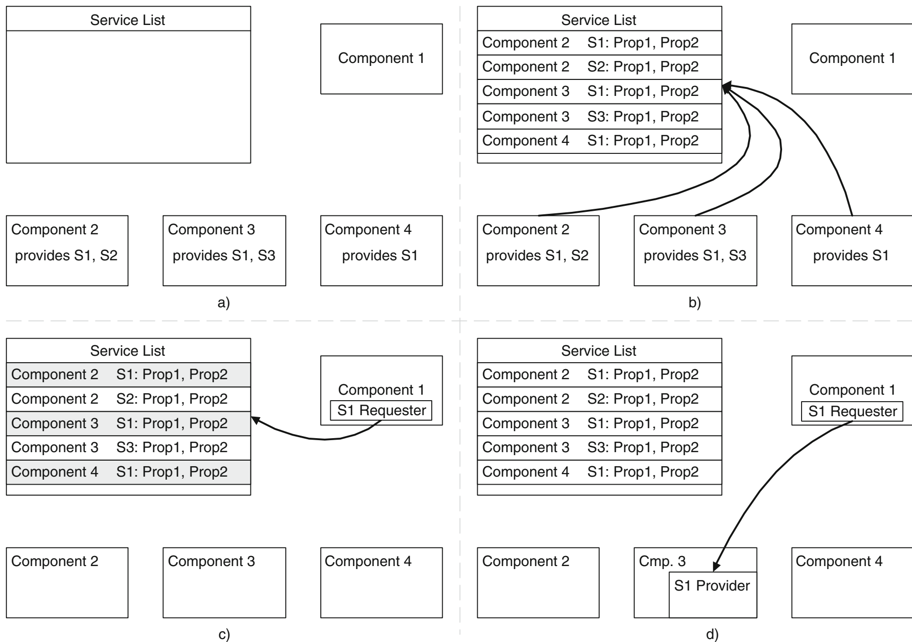


Fig. 2. a) A system state before any services are offered. b) The components offered their services with current properties. c) The service requester asks for services with specific properties. d) The service requester establishes a link to the selected service provider.

is proposed. Work in the ActIPret project has shown that formalising vision functions to this extent is already cumbersome. It also indicates that the science of computer vision did not yet settle enough to reward work in this area.

cognitive vision functions detection, tracking and recognition.

3.7 Implementation: `zwork` and Its Graphical User Interface (GUI)

The software implementation of the framework is based on RPC (Remote Procedure Call) under the operating system Linux. RPC has been chosen because it is very established, standardised, light weight, fast and easy to use. CORBA as an alternative networking layer imposes a larger overhead and is generally tailored to other uses, such as web-services rather than a real-time vision systems. Moreover CORBA is still developing and different implementations are not interchangeable.

The framework implementation is referred to as `zwork`. It provides the communication mechanisms to establish and request services. Deliverable D1.3, available from the ActIPret home-page (actipret.infa.tuwien.ac.at), gives more details about `zwork`, an installation and a programmer's guide.

In order to control such a framework and give the user, and especially the developer, feedback of the results of various components, a graphical user interface

(GUI) is mandatory. Results of different vision modules should be superimposed on the original image to indicate correct operation at a glance. A GUI with the capability to display text and drawing primitives as an overlay of the processed 2D images and a 3D graphical display was implemented, see Fig. 3. It provides the following functions:

– Display of the original 2D images from the cameras together with an overlay of vision processing results. A button just below the image selects this alternate display.
– Display of 3D representation of the objects (see Figure 4 for an example), trajectories and other spatial information. The three sliders to the left, right and below the image allow zoom in and spherical angle change for the view of the 3D scene.
– Display all services available at present (box to the left top).
– Display of all components (box to the left centre) and currently running services. Graphical output for each component can be enabled or disabled.
– Display of textual information (box to the bottom right) to output symbolic information, e.g., recognised gesture or activity concepts.
– Various functions for debugging and presentation (bottom left), eg. for logging data and images, and to make a snap shot of the GUI (such as presented in Fig. 3).

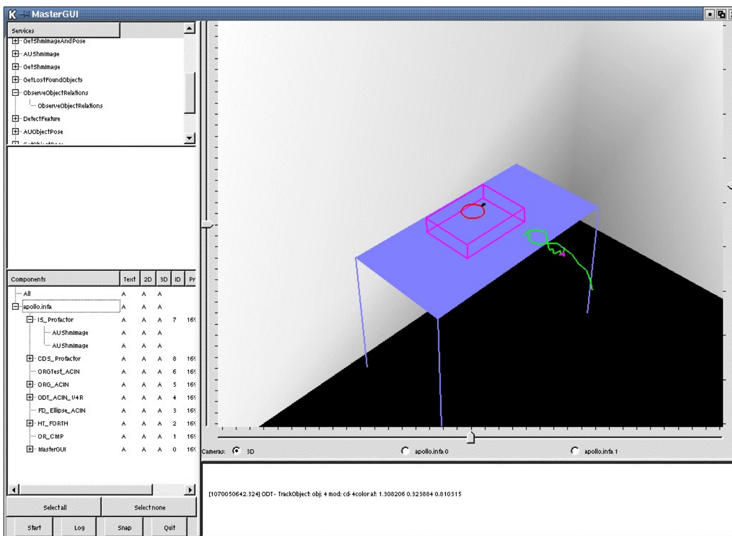


Fig. 3. GUI of the Cognitive Vision Framework developed in ActIPret. In the 3D display (top left), the green (bright) trajectory is the 3D hand trajectory. The ellipses are the two CDs recognised, also marked with an “f” for found. This 3D rendering can be viewed from different angles using the sliders in the GUI.

4 Experiments in the ActIPret Project

Fig. 4 gives on overview of the ActIPret Demonstrator architecture. The system was rigorously tested by the ActIPret partners in an integration meeting at a partners site (Fig. 5). For these tests, there were no Activity Planner and User HMI components available. As a result the internal ARE (Activity Reasoning Engine, see Fig. 4) output was displayed at the GUI.

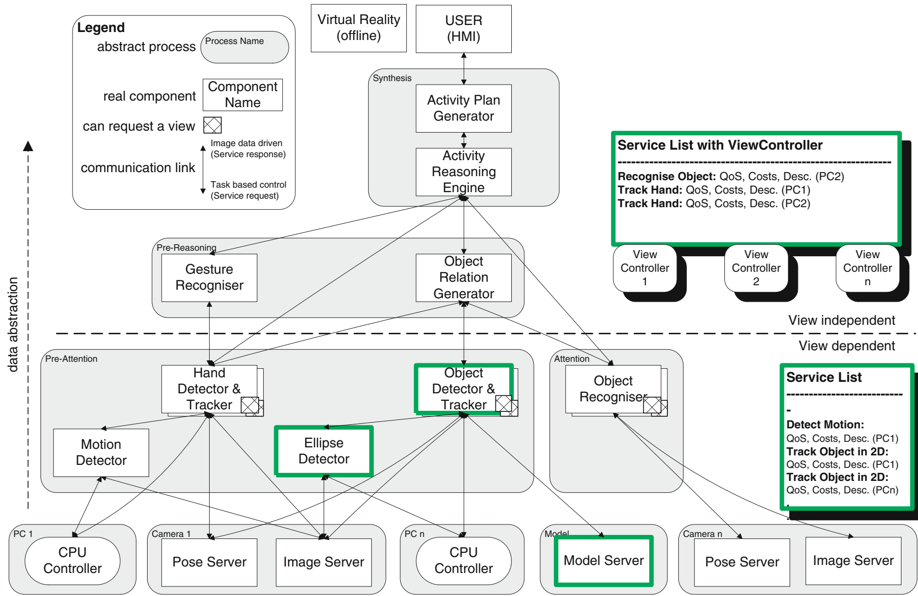


Fig. 4. The components of the ActIPret demonstration system

Of particular interest is the study of the interaction between components. Two examples are given. The first example is the automatic initialisation of tracking using detection. The second is the contextual use of hand tracking to limit the area to search for potential objects that might be grasped.

Ellipse detection is used to initialise ellipse tracking. Fig. 6 shows this coupling of two components for a lamp. The technique of detection is based on a hierarchical grouping method that is very efficient in computing [24]. By exploiting an ordering according to most likely ellipse arcs, arc groupings and, finally, ellipses, it also produces reliable results. Computing time is less than 500 ms on an Athlon 1.880+ PC for the full image (768x576 pixels) and only 50 ms when used in a ROI (Region of Interest) provided by hand tracking (exemplified in the second example). Tracking exploits the technique described in detail in [22]. In the Fig. 6 only the first frame is shown.

An example of using context in ActIPret is the exploitation of the hand trajectory to narrow the SOI (Space of Interest) for computational expensive components like object detection and object recognition. Fig. 7 is an instant where

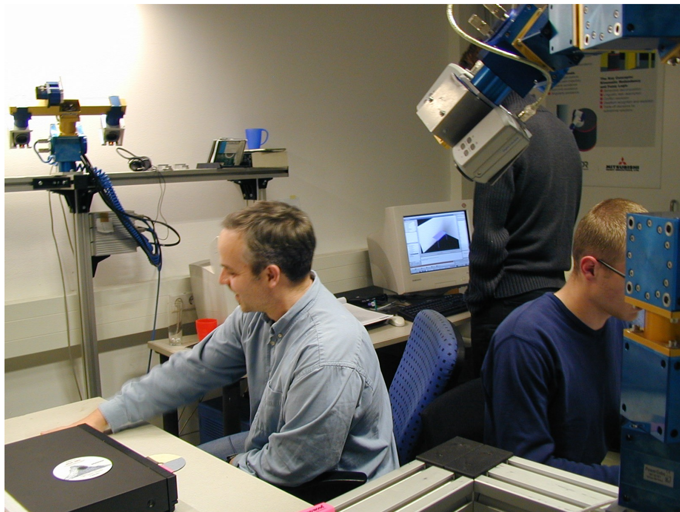


Fig. 5. A live demonstration of the ActIPret framework in the laboratory at PROFACTOR. Top left is the active fixed stereo system, top right the stereo system on the robot arm. The screen in the background shows the GUI of the framework while Jon Howell (from the ActIPret partners from the University of Sussex) is placing a CD in the player.

the hand is tracked (ellipses around the hand) and the motion direction indicates the image region where potential grasp objects, in this case CDs, might be detected. The search region on the top of the CD-player is caused by a previous hand motion. The ellipses reported are generated using the method of hierarchical grouping presented in [24]. It can be seen that the most likely hypothesis, marked with "0", is the CD in both squares.

In the first example, the Object Detector & Tracker (ODT, please refer to Fig. 4) needs external help to provide the promised service of detection and calls the Ellipse Detector (ED) to detect the lamp object. In the second example it is the Object Relationship Generator (ORG) that establishes the context between Hand Detector & Tracker (HDT) and ODT. The ODT then again selects the service from an ED to initialise tracking. In this case a second option is available, where the service of Object Recognition (OR) could provide object locations for tracking. This is investigated with partner CMP (Center for Machine Perception, Czech Technical University, Prague) within the ActIPret project. It shows that with a growing number of components, more versatile systems can be built and that there are several routes to fulfil a task, which adds to the robustness of the overall system. This later aspect is the ultimate driving force to build the framework and it is seen as a promising method to approach building widely applicable CVS.

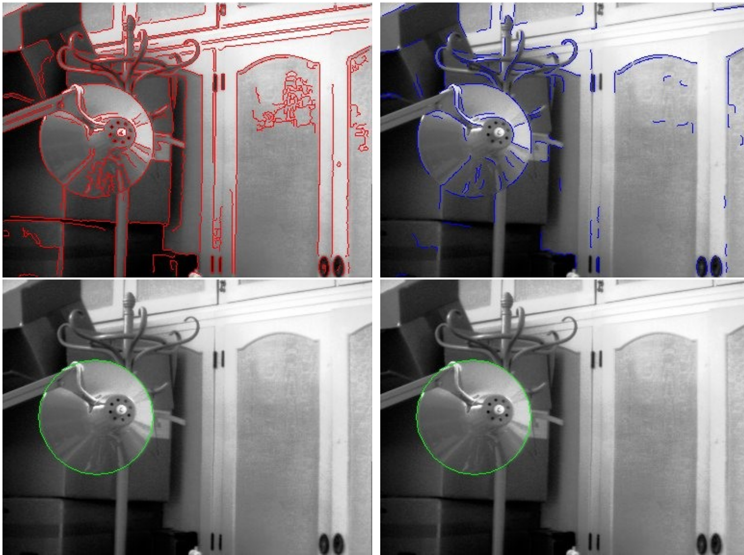


Fig. 6. Detecting all ellipses in the scene to track the object. Top left shows the edges, top right the grouped arcs, bottom left the only detected ellipse and the final image the first step of tracking using the technique of [22].

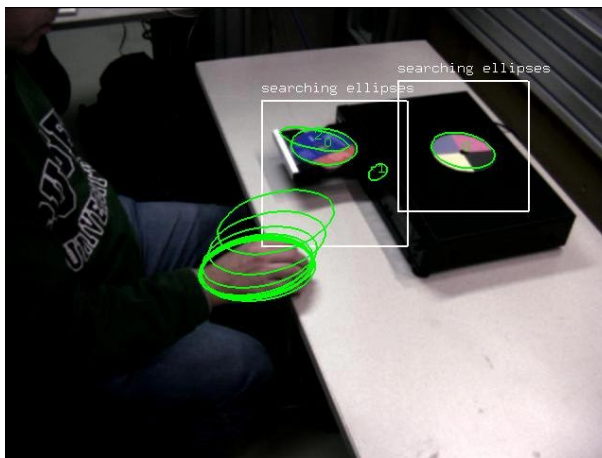


Fig. 7. Display of information from Hand Tracker and Ellipse Detector overlaid on the original image. The green ellipses represent the hand tracked over the last images. The two squares contain information on ellipse detection within this ROI. Each time the ellipses detected are displayed and the number gives the ranking, with number 0 being the highest ranking and indicating, the both CDs have been found correctly.

5 Conclusions

The intention of this paper is to raise the awareness that building CVS includes many facets of science for fulfilling the tasks intended. The argument is that embodiment, multiple modes of sensing with vision as a powerful yet demanding sense (in terms of resources as well as methods) and reasoning about the percepts as well as the situation and goals, call for a supporting software framework.

The paper then outlined one tool, `zwork`, to provide such a framework for the ActIPret project. This framework will be further used and improved in the ActIPret project. The main project goal is to study the interaction of vision functionalities for activity interpretation. The central theme of work is to use contextual information to increase the robustness of the interpretation. The framework is the necessary tool to do so easily. `zwork` operates with a quick component start-up, a data transfer time of one millisecond and it has, due to its simple design, a short learning phase for the vision researcher. The developers plan to make the framework tool `zwork` publicly available. Please contact the authors.

The notion of QoS was introduced to formalise the properties of vision functionalities encapsulated in the components. While recent work on performance evaluation started to aid such a formalisation, computer vision methods elucidate such a scientific formalisation far more than methods in other engineering sciences. It is therefore thought essential to work along this line to arrive at clearly defined properties of methods for given context and use. The component-based approach enforces this development and the advantages gained from building components (reuse in other applications, easy use of methods by other, formal comparison of methods) will have the effect to seriously evaluate the advance of vision methods.

The current demonstrations work with intermediate functionality of the components. While simple options for redundant components (finding CDs either with recognition or ellipse detection) have been realised, the full power of a component-based approach becomes visible if *many* components offer similar or competing services. Hence, we want to encourage other vision researchers to exchange their developments and components with our framework to build more powerful and versatile Cognitive Vision Systems.

References

1. Albus, J.S.: 4-D/RCS: A reference model architecture for Demo III. In: IEEE ISIC/CIRA/ISAS Joint Conf. (September 1998)
2. Arkin, R.C.: Behaviour based robotics. The MIT Press (1998)
3. Blum, S.: OSCAR - Eine Systemarchitektur für den autonomen, mobilen Roboter MARVIN. In: Autonome Mobile Systeme, Informatik Aktuell, pp. 218–230 (November 2000)
4. Crowley, J.L.: Integration and Control of Reactive Visual Processes. Robotics and Autonomous Systems 15(1) (1995)
5. Crowley, J., Christensen, H. (eds.): Vision as Process. Springer (1995)
6. University of surrey. Recognition and Vision Library (2003), `http://ravl.sourceforge.net`

7. Fink, G.A., Jungclaus, N., Kummert, F., Ritter, H., Sagerer, G.: A distributed system for integrated speech and image understanding. In: International Symposium on Artificial Intelligence, Cancun, Mexico, pp. 117–126 (1996)
8. Klusch, M., Sycara, K.: Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In: Omicini, A., et al. (eds.) Coordination of Internet Agents. Springer (2001)
9. Klusch, M., Bergamaschi, S., Edwards, P., Petta, P. (eds.): Intelligent Information Agents. LNCS (LNAI), vol. 2586. Springer, Heidelberg (2003)
10. Konstantinides, K., Rasure, J.R.: The Khoros software development environment for image and signal processing. IEEE Transactions on Image Processing 3(3), 243–252 (1994)
11. Lindstöm, M., Orebäck, A., Christensen, H.: Berra: research architecture for service robots. In: IEEE ICRA, pp. 3278–3283 (2000)
12. Object Management Group (OMG). CORBAservices: Common Object Service Specification (March 1995)
13. Prouskas, K., Pitt, J.: A real-time architecture for time-aware agents. In: IEEE SMC (2003) (in press)
14. Schlegel, C., Wörz, R.: The Software Framework SmartSoft for Implementing Sensorimotor Systems. In: IEEE/RSJ IROS 1999, Kyongju, Korea, pp. 1610–1616 (October 1999)
15. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.: Ros: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
16. Hawes, N., Wyatt, J.: Engineering Intelligent Information-Processing Systems with CAST. Advanced Engineering Infomatics 24(1), 27–29 (2010)
17. Shoham, Y.: What we talk about when we talk about software agents. IEEE Intelligent Systems 14(2), 28–31 (1999)
18. Ponweiser, W., Umgeher, G., Vincze, M.: A Reusable Dynamic Framework for Cognitive Vision Systems. In: Workshop on Computer Vision System Control Architectures at ICVS 2003, Graz, pp. 31–34 (2003)
19. Szyperski, C., Pfister, C.: Workshop on Component-Oriented Programming, Summary. In: Mühlhäuser M. (ed.) Special Issues in Object-Oriented Programming - ECOOP 1996 Workshop Reader. dpunkt Verlag, Heidelberg (1996)
20. Szyperski, C.: Component software. Addison Wesley, United Kingdom (1999)
21. Veryard, R.: Component-Based Business: Plug and Play. Springer (2001)
22. Vincze, M., Ayromlou, M., Ponweiser, W., Zillich, M.: Edge Projected Integration of Image and Model Cues for Robust Model-Based Object Tracking. Int. Journal of Robotics Research 20(7), 533–552 (2001)
23. Wooldridge, M., Jennings, N.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10(2) (1995)
24. Zillich, M., Matas, J.: Ellipse Detection using Efficient Grouping of Arc Segments. In: 27th Workshop of the Austrian Association of Pattern Recognition OAGM/AAPR, pp. 143–148 (2003)

KOROS* Initiative: Automatized Throwing and Catching for Material Transportation

Martin Pongratz[1], Klaus Pollhammer[1], and Alexander Szep[2]

[1] Vienna University of Technology, Institute of Computer Technology
Gußhausstraße. 27-29 / 384, 1040 Wien, Austria
http://www.ict.tuwien.ac.at
[2] Vienna University of Technology, Automation and Control Institute
Gußhausstraße. 27-29 / 376, 1040 Wien, Austria
{pongratz,pollhammer}@ict.tuwien.ac.at, alexander.szep@tuwien.ac.at
http://www.acin.tuwien.ac.at

Abstract. Catching a thrown object has increasingly been a subject of research. The reason has largely been to demonstrate the advances in robot technology. Besides this academic usage also the application of throwing and catching for material transport has been proposed. Within the KOROS initiative at the Vienna University of Technology the transport-by-throwing approach will be developed further. Based on multiple cameras and advanced robotic arms a practical evaluation of the approach will be done. The realization with state of the art equipment will enable to identify possible fields of application as well as current limitations of the transport-by-throwing approach. Especially soft throwing and catching, exposing the transported objects to minimal forces, are of main interest.

Keywords: KOROS, versatile manufacturing, flexible transportation, transport-by-throwing, robotics, catching, throwing.

1 Introduction

State of the art production faces the challenge of the conflicting requirements of automatization and flexibility [1]. The main driver for this challenge is the raising demand for individual products and the increasing number of product's variants over the last decades. This circumstance has raised the requirements for production facilities and automatization solutions. Versatile solutions for the material flow, that are able to cope with unexpected situations, are required

* The tranport-by-throwing scenario presented here is one of the activities within the KOROS (collaborating robotic systems) initiative of the Vienna University of Technology. The initiative has be established in 2010 after a successful proposal for robotic infrastructure including a 3D-display, two KUKA LWR 4+ arms and the world-wide first Alebaran Romeo humanoid robot (panned to be delivered in early 2013). The funding for the infrastructure has been provided by the WWTF as well as the Vienna University of Technology.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 136–143, 2012.
© Springer-Verlag Berlin Heidelberg 2012

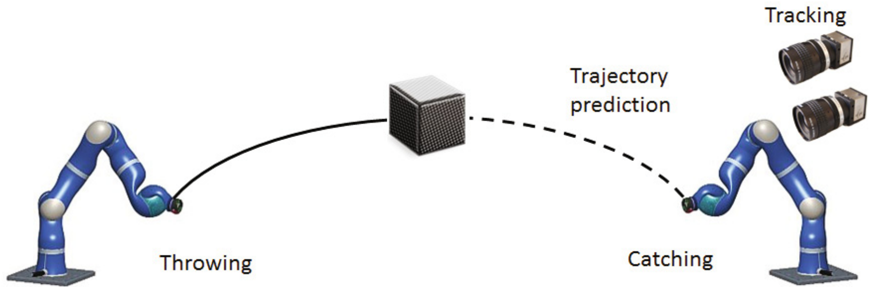


Fig. 1. Overview of the transport-by-throwing approach with four functional divisions

[1]. The concurrent increased usage of robots [2] for handling tasks opens the opportunity to use the robots also to enable new flexible ways of material flow.

Transport-by-throwing has been proposed by Frank [3]. The basic principle of automatized throwing and catching significantly increases the flexibility due to the laps of additional infrastructure for material handling like conveyor belts. Regarding the three main shapes of flexibility, namely layout-flexibility, throughput-flexibility and the flexibility of the transported material proposed by Günthner [4], the transport-by-throwing approach has excellent properties regarding the layout-flexibility and the throughput-flexibility. In case the production line is composed of modular production units the layout of the whole line is only restricted by the range of throwing. In case of longer transportation paths routing via other production units is possible (similar to multi-hop routing in communication networks). Basically all other (modular) production units can be used for transporting in case they are within the reach of the origin unit. In addition to transportation on the same height-level also the transfer to different levels is possible without additional infrastructure. In terms of the flexibility of transported material, research has only dealt with highly symmetrical (point-symmetrical, axial-symmetrical) objects like tennis balls, cylinders or similar objects [5] [6] [7] [8] [9].

Currently the field of application of transport-by-throwing is seen in the area of objects with a mass of up to 100 g for sorting and separation [7]. One of the future research scenarios presented in section 3 will deal with the identification of suitable objects for this transport approach.

Based on initial results examining transportation over a distance of 3 m with a initial velocity of approximately 10 m/s the current work is based on slower thrown objects (5 m/s) and a transportation distance of 2.5 m. This combination represents the minimal velocity required for a tennis ball to travel the distance of 2.5 m [10], thus limiting the velocity requirements on throwing and catching devices.

The whole process of throwing and catching can be divided into four major functions [3]. Figure 1 illustrates a throwing and catching system and outlines the four fundamental functions. The breakdown into four functional divisions is

based on the main functions throwing and catching as well as the derived requirements for pose (position and orientation) prediction and tracking. Pose prediction is necessary in order to gain information about the optimal interception position and suitable deceleration trajectories at the catching site. Additional tracking is necessary because the trajectory of the thrown object is highly sensitive to small, unavoidable deviations in the launching parameters as well as other factors (e.g. air streams, deviations in the object's physical properties) [3] [9]. The tracking information is used to refine the pose prediction during the flight. Regarding the prior work based on point symmetrical object's the requirement for orientation-prediction and -tracking has been omitted.

Throwing. During throwing the object is accelerated up to the velocity that is required to reach the catching area of the destination unit. The main parameter of the throwing process is the launching velocity. Both magnitude and direction of the velocity are highly important as the trajectory is highly sensitive to deviations in the launching velocity [3]. Other important properties of the throwing process are the acceleration trajectory and the forces during the acceleration. The acceleration trajectory is responsible for the rotational behavior of the object throughout the flight. Linear acceleration units have been used in order to enable accurate throwing with a stable object orientation [11]. The maximum force, the object is exposed to, is responsible for deforming of the object or damage and has to be limited depending on the object's properties.

Catching. The task of catching deals with the controlled deceleration of the object at the destination unit. Information about the optimal interception position is based on the initial launching parameters as well as the tracking information. In order to minimize the forces on the object during catching the deceleration trajectory has to initially follow the object's trajectory. After the thrown object is fixed on the catching device (e.g. actively grasped, passively fixed by friction) the deceleration trajectory may change to a different shape. This deviation also results in forces on the caught object, that have to be considered as well. A possible catching trajectory is a circle that tangents the object's trajectory in the interception point and afterwards decelerates the object with a circular movement.

Prediction. Pose prediction is necessary to enable catching at the destination unit. The information of the predicted trajectory and orientation is used to position the catching device at the optimal interception position prior to the object reaching this point. Also the aligning of the object's and the catching device's velocity is based on the prediction system. The information the prediction system is based on are the initial launching parameters as well as the information from the tracking system. Prediction can be based on physical models, modeling the flight properties of the object [6] [9], generic curve fitting [9] or approaches based on learning [12].

Tracking. Tracking of a thrown object is mainly based on video information due to the high dynamics of the movement. Approaches based on single cameras

[6], a set of two cameras [9] [13] as well as multiple cameras [14] [15] are existing. Depending on the object's symmetrical properties the image processing's complexity varies, ranging from simple object detection to pose estimation and tracking.

2 Related Work

In addition to the transport-related application of throwing and catching also the usage of this application for demonstrations of the abilities of robots has been done. Catching emerging from grasping has been surveyed at the University of Tokyo [16] [17] [18] [19]. A three-fingered hand was developed and first used to grip soft objects (foam ball, foam cylinder). A specially developed low-resolution high-speed vision system working at 1000 frames per second is used to determine the position of the object. This specially designed vision system has a limited range of view and uses only binary image information which is equal to a black/white image. The algorithm designed for the tracking of the object relies on the high frame rate of the visual information (1000 frames per second) and the resulting small movement of the object between two captured frames. Usage of the so-called "self windowing" algorithm is possible in combination with a tracking algorithm for the sensor system that keeps the object under investigation in the center of the image. This algorithm is not comparable to other object tracking algorithms in machine vision, because it was purposely built for this application [13]. Results achieved (in order of publication date) are: catching of a foam ball [18], catching of a foam cylinder [18], dribbling a ball on a flat surface [20] and dynamic regrasping [21] on a small range. More recent research has focused on knotting and object handling in hand. Common attributes of these topics are the small range the object moves before it is caught and the high-speed position detection. These two restrictions allow catching based on the actual measured position of the object, omitting the need of an interception-time, position and orientation prediction system.

Approaches for throwing and catching including trajectory estimation and trajectory prediction were already done in 1995 by Hong and Slotine at the Massachusetts Institute of Technology [22]. Using two cameras they were able to successfully complete "trajectory matching" of a thrown ball in 70 to 80 % of all throws. The trajectory fitted to the measured positions was a simple parabolic function. In 1997 Hove and Slotine were able to show the realization of the whole system based on a set of more advanced cameras [23]. At that time the rate of successfully caught balls, thrown to the robot's workspace from random positions, was about 75 % as well. The slow closure of the catching device as well as the small allowed timing error were the main challenges they had to face.

Another approach done by Frese and colleagues of the Institute of Robotics and Mechatronics at the German Aerospace Center in 2001 used an extended Kalman filter for trajectory prediction while the position acquisition was based on a stereo vision camera system. The prediction accuracy was better than 100 mm even for the early flight phase [24]. Further research at the same institution

from Bäuml et al. [14] presents three different catching strategies "soft", "latest" and "cool" that attribute the movement of the robotic arm. The setup is based on a DLR LWR 3 arm (prototype series, predecessor of the KUKA LWR 4 arm) and the same visual tracking and Extended Kalman filter based prediction as in 2001. Tracking is processed on a quad-core processor while the catching strategy related calculations are done on a 32-core cluster. The catch success rate is higher than 80 %. Common for the strategies is the attribute that the ball is caught in a "hard" way which means that the arm is not moving along the trajectory of the ball during catching. This results in mayor forces or torques on the arm and the caught object during catching. More recent work deals with the usage of a rolling humanoid robot, outfitted with two arms, for ball catching and the preparation of coffee. A catch rate of 80 % is achieved for the robot system as well, considering balls within the catch space of the robot [15].

3 Planned Research Topics

Based on the robotic hardware of the KOROS work group the transport-by-throwing approach will be developed further. The findings in previous research at the Vienna University of Technology based on a coil-based throwing device, a tennis ball and an impact position detection system [9] will be used as the startingpoint. Four research topics will be outlined in the next subsections.

3.1 Throwing and Catching of Point-Symmetrical Objects

The first scenario is the realization of previous established algorithms [9] with robotic arms, enabling the transportation of a tennis ball between two robotic arms based on throwing and catching. For throwing an arm outfitted with a cup will be used while the catching device will be another robotic arm outfitted with a baseball glove. Initially the "hard" ball catching will be used but the progression towards "soft" catching is planned. Here "hard" catching is defined as catching without specific movement of the robotic arm along the object's trajectory while "soft" catching is defined as catching with matched trajectories of object and robotic arm, resulting in smaller forces on the objects and the robotic arm. Due to the relatively high velocity of the ball (concerning the maximum velocity of the end-effector of the KUKA LWR 4 arm) optimal usage of the robot's kinematic is planned. While related work [14] is based on a vertical axis of the first joint, rotation of this axis into a horizontal position will enable faster movements of the end-effector along the ball's trajectory.

3.2 Practical Experiments on Object's Flight Properties

A basic step in application of the transport-by-throwing approach to industrial applications is the identification of suitable objects for throwing and catching. The flight properties of the objects are a main criteria, besides the mechanical properties, influencing the flight trajectory. Based on a simulation and practical experiments the impact of size, mass and shape of the object on the flight properties will

be examined. Keeping two of the three factors constant while varying the remaining will establish information to identify suitable objects on a broader range.

3.3 Evaluation of Throwing Strategies

The high number of degrees of freedom (7 DoFs) of the KUKA LWR 4 arm allows to accelerate an object to the same launching velocity on different trajectories. These different trajectories result in different rotational velocities of the object in the throwing instant and thus also influence the stability of the object's orientation during the flight. In interaction with the experiments on the object's flight properties also the optimized acceleration trajectory for an object will be examined. Based on a set of practical experiments the goal is again to derive general rules for the acceleration trajectory to enable a stable flight of the object.

3.4 Experience Based Trajectory Prediction

Most in 2 presented approaches predict the object's motion based on physical models or based on a Kalman filters. The goal for the experience based trajectory prediction is to predict the object's motion based on a set of reference throws. Currently two strategies for processing the position information are planned: one using a quasi-continuous spatial information, the other based on quantized spatial information. In the second case the object's movement will be mapped from the continuous spatial space into a quantized spatial space, reducing the numbers of ingestible positions for the object. This step is comparable to the digitalization of audio data on a compact disk. While the quantization introduces an additional error, the effect of this error will be minimized through the alignment of the quantization intervals to the position detection system's accuracy. Figure 2 shows a example of objects positions in continuous spatial space and quantized spatial space (size of cuboids or quantization intervals: $q_x = 30mm$, $q_y = 30mm$, $q_z = 50mm$). Based on the reduced number of ingestible positions the information about the trajectories will be stored as experience. This experience will be used to find similar throws to previously experienced throws and predict the current trajectory

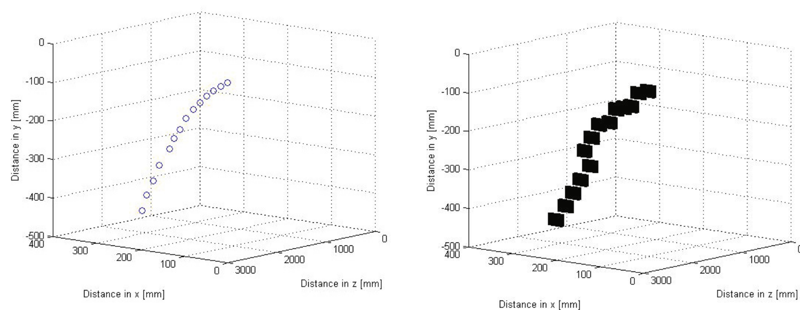


Fig. 2. Visualization of positions in continuous spatial diagram (left side, circles) and quantized spatial space (right diagram, cuboids)

on this basis. Advantages and disadvantages of both methods will be evaluated in terms of computation expense and prediction accuracy.

4 Conclusion

The infrastructure of the KOROS work group at the Vienna University of Technology is an excellent basement for further research of the transport-by-throwing approach. Several scenarios were outlined in this overview and it is hoped that the KOROS initiative will help to revolutionize the material-transport for small-scale items.

Acknowledgments. The would like to thank the Vienna Science and Technology Fund and the Vienna University of Technology for their support that permitted the acquisition of the KOROS infrastructure.

References

1. BVL Bundesvereinigung Logistik, Trends und Strategien in der Logistik - Ein Blick auf die Agenda des Managements 2010, Deutscher Verkehrs-Verlag GmbH, Bremen (2005) ISBN 3-87154-331-4
2. Quest Trend Magzin Online, `http://www.quest-trendmagazin.de/Einsatz-von-Robotern-steigt-20.174.0.html?&L=0` (last visit August 23, 2011)
3. Frank, H., Wellerdick-Wojtasik, N., Hagebeuker, B., Novak, G., Mahlknecht, S.: Throwing Objects – A bio-inspired Approach for the Transportation of Parts. In: IEEE International Conference on Robotics and Biomimetics, ROBIO 2006, December 17-20, pp. 91–96 (2006)
4. Günthner, W., Heinecker, M.: Modulare Materialflusssysteme für wandelbare Fabrikstrukturen - Bewertungs- und Gestaltungsrichtlinien für wandelbare Materialflusssysteme, Internetplattform Logistics.de, Erscheinungsdatum, May 19 (2006)
5. Frank, H., Barteit, D., Meyer, M., Mittnacht, A., Novak, G., Mahlknech, S.: Optimized Control Methods for Capturing Flying Objects with a Cartesian Robot. In: Proceedings on 3rd IEEE International Conference on Robotics, Automation and Mechatronics, Chengdu, China, September 22 - 24 (2008)
6. Barteit, D., Frank, H., Kupzog, F.: Accurate prediction of interception positions for catching thrown objects in production systems. In: Proceedings on 6th IEEE International Conference on Industrial Informatics, Daejeon, Korea, July 13 - 16 (2008)
7. Frank, H., Mittnacht, A., Scheiermann, J.: Throwing of Cylinder Shaped Objects. In: Proceedings on 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2009), Singapore, July 14-17, pp. 59–64 (2009)
8. Barteit, D., Frank, H., Pongratz, M., Kupzog, F.: Measuring the Intersection of a Thrown Object with a Vertical Plane. Paper is accepted for 7 th IEEE International Conference on Industrial Informatics (INDIN 2009), Cardiff, UK, June 24 -26 (2009)
9. Pongratz, M., Kupzog, F., Frank, H., Barteit, D.: Transport by Throwing - A bio-inspired Approach. In: Proceedings on 8th IEEE International Conference on Industrial Informatics, Osaka, Japan, July 13 - 16, pp. 685–689 (2010)

10. Animation - Der schräge/schiefe Wurf ohne und mit Luftwiderstand/dynamischem Auftrieb/Magnus-Effekt, http://www.tutz.ws/JS/Simulation-Schraeger-Wurf-F_L-F_A-F_M.html (last visit August 25, 2011)
11. Frank, H., Mittnacht, A., Scheiermann, J.: "Throwing of cylinder-shaped objects. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2009, July 14-17, pp. 59–64 (2009)
12. Mulling, K., Kober, J., Peters, J.: A biomimetic approach to robot table tennis. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 18-22, pp. 1921–1926 (2010)
13. Ishii, I., Nakabo, Y., Ishikawa, M.: Target tracking algorithm for 1 ms visual feedback system using massively parallel processing. In: Proceedings of IEEE International Conference on Robotics and Automation, April 22-28, vol. 3, pp. 2309–2314 (1996)
14. Bäuml, B., Wimböck, T., Hirzinger, G.: Kinematically Optimal Catching a Flying Ball with a Hand-Arm-System (2010)
15. Bäuml, B., Schmidt, F., Wimböck, T., Birbach, O., Dietrich, A., Fuchs, M., Friedl, W., Frese, U., Borst, C., Grebenstein, M., Eiberger, O., Hirzinger, G.: Catching Flying Balls and Preparing Coffee: Humanoid Rollin'Justin Performs Dynamic and Sensitive tasks (2011)
16. Namiki, A., Nakabo, Y., Ishii, I., Ishikawa, M.: High speed grasping using visual and force feedback. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 4, pp. 3195–3200 (1999)
17. Namiki, A., Ishikawa, M.: Robotic catching using a direct mapping from visual information to motor command. In: Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2003, September 14-19, vol. 2, pp. 2400–2405 (2003)
18. Namiki, A., Imai, Y., Ishikawa, M., Kaneko, M.: Development of a high-speed multifingered hand system and its application to catching. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), October 27-31, vol. 3, pp. 2666–2671 (2003)
19. Imai, Y., Namiki, A., Hashimoto, K., Ishikawa, M.: Dynamic active catching using a high-speed multifingered hand and a high-speed vision system. In: Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2004, April 26-May 1, vol. 2, pp. 1849–1854 (2004)
20. Shiokata, D., Namiki, A., Ishikawa, M.: Robot dribbling using a high-speed multifingered hand and a high-speed vision system. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), August 2-6, pp. 2097–2102 (2005)
21. Furukawa, N., Namiki, A., Taku, S., Ishikawa, M.: Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In: Proceedings of 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, pp. 181–187 (2006)
22. Hong, W., Slotine, J.-J.E.: Experiments in Hand-Eye Coordination Using Active Vision. In: Khatib, O., Salisbury, J.K. (eds.) Experimental Robotics IV. LNCIS, vol. 223, pp. 130–139. Springer, Heidelberg (1995)
23. Hove, B., Slotine, J.-J.E.: Experiments in Robotic Catching. In: American Control Conference, June 26-28, pp. 380–386 (1991)
24. Frese, U., Bauml, B., Haidacher, S., Schreiber, G., Schaefer, I., Hahnle, M., Hirzinger, G.: Off-the-shelf vision for a robotic ball catcher. In: Proceedings of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 1623–1629 (2001)

Cognitive Decision Unit Applied to Autonomous Robots

Dietmar Bruckner and Friedrich Gelbard

Institute of Computer Technology, Vienna University of Technology,
Gusshausstrasse 27/384, 1040 Vienna, Austria, Europe
{bruckner,gelbard}@ict.tuwien.ac.at

Abstract. A novel approach to equip robots with human-like capabilities is to use meta-psychology – the theoretic foundation of psychoanalysis. We used meta-psychology as archetype for a decision making framework to control a robot. This has been achieved recently in theory and in simulation. However, when moving to a real robotic platform, additional things have to be considered. In this article we show how to fill the gap between sensing, environmental interaction, and decision making by grounding these topics with an agent's internal needs using the concepts of meta-psychology. The use of the common humanoid robot platform NAO compelled us to deal with complex situations and disturbed sensor readings. An implemented visual marker detecting system helps to detect objects in the surrounding environment, representing energy sources. We show how it is possible to use the psychoanalytically inspired framework ARS to control a real world application, the robot NAO.

Keywords: Cognitive Automation, Robotic Control Unit, Simulation, Sensor Data Interpretation.

1 Introduction

As of today, artificial intelligence algorithms like vacuum cleaning robots or chess playing computers [1] fulfill highly specialized tasks. A more complex approach to artificial intelligence is represented by a new research community called Artificial General Intelligence (AGI).

The Artificial Recognition System (ARS) project develops such an architecture. Ten years ago, when this project started, the motivational question was how to extract meaning from data provided by hundreds of thousands of sensor nodes each second [2]. A possibility to solve this problem is to model mechanisms of the human mind. A novel concept is to use the theoretical background of psychoanalysis as foundation [3]. The resulting model is described in depth in [4-6] and a brief overview is given in Section 3.

The first application used to evaluate the model was a kitchen equipped with various types of sensors [7]. In the course of the project, focus shifted to a simulated world of artificial life [8]. A first attempt to use the psychoanalytically inspired control system to control a robot is mentioned in [9]. The used robot was Tinyphoon [10]: a small, two-wheeled, cube shaped soccer robot equipped with various sensors.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336 , pp. 144–149, 2012.
© Springer-Verlag Berlin Heidelberg 2012

This approach was limited, as a rich control system was applied to a robot with only two wheels as actuators. This violated the demand of ecological balance formulated in the design principles for embodied agents [11].

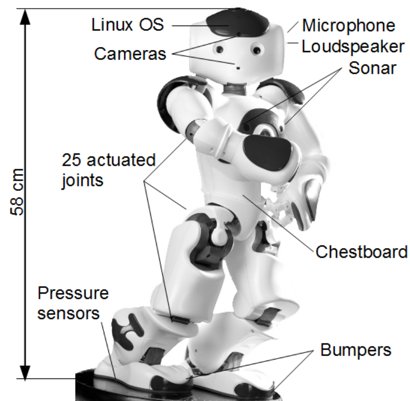


Fig. 1. NAO Robot

In contrast, humanoid robots. offer a wide range of actuators and sensors together with more interesting body features like motor temperature, joint sensors, and power supply. This article aims at presenting the application of a psychoanalytically inspired decision unit to such a robot. The chosen platform is NAO – a standard robot platform (see Fig. 1).

The full report of this work has been published earlier [12].

2 State of the Art

2.1 Mobile Platforms

Commonly used mobile platforms for research purposes are the wheel driven pioneer robots. They are produced by Mobile Robots and can be shipped in different configurations due to their modular design. The basic configuration of pioneer robots is a mobile platform without sensors and actuators. Recent developments in RoboCup show practical applications of pioneer robots and demonstrate their capabilities.

Another mobile platform architecture is PR2. PR2 is manufactured by Willow Garage. PR2 has two arms and wheel drive and is able to grasp and transport objects [13]. The disadvantage of PR2 is its high price.

For both platforms, pioneer robots as well as PR2, the open-source meta-operating system ROS [14] can be used. ROS offers a standardized API to control mobile platforms.

But for the current work we used NAO (Fig. 1). NAO offers human-like arms and legs and provides a thoroughly documented API.

2.2 Control Architectures

Widely used control architectures for autonomous robots are the subsumption architecture [15] and the belief-desire-intention (BDI) architecture [16]. The subsumption architecture is able to fulfill simple tasks like following a line or collecting certain things. For more difficult tasks the subsumption architecture is rarely used.

In contrast to the subsumption architecture, BDI architecture can easily be adapted to altered tasks. For a different task, only the database of the BDI architecture needs to be changed; the program code remains the same. The disadvantage of BDI is the lack of planning and learning abilities.

More specific control architectures are cognitive architectures like SOAR [17] and ACT-R [18]. These control architectures model human cognition. Common characteristics of cognitive architectures are specific memory structures and categorization of knowledge. Often, ontologies are used and an inference engine based on rules and facts is implemented.

In contrast to artificial intelligence (AI) which is focused on specific simple tasks, artificial general intelligence (AGI) intends to solve complex problems similar to human's intelligence. AGI tries to reconstruct human thinking processes and tries to develop models of the human mind.

One important AGI project is LIDA [19]. LIDA processes perceptions in cognitive cycles. One cognitive cycle is subdivided in understanding phase, consciousness phase, and action selection phase.

A second project is OpenCog [20]. Specific memory structures and cognitive processes are the characteristics of OpenCog. Applications are virtual pets and child learning simulations.

3 Model

The aim of the ARS-project is to implement psychoanalytic notions in artificial general intelligence and to demonstrate the feasibility in a simulation model. One important aspect of ARS is the clear interface which connects ARS to its environment. Incoming data are perceptions from the environment, bodily drives, and homeostatic sensor data. Outgoing data are the actuator control commands. Fig. 2 shows the basic concept of ARS with its interfaces.

In Fig. 2 Id represents the perceptions, bodily drives, and homeostatic sensor data. On the other hand, Super-Ego represents social rules like: "Share your food with others." or "Be polite and not aggressive."

If the demands of Id are conflicting with social rules and reality, Ego needs to interfere and mediate between Super-Ego and Id. Ego uses the defense mechanisms to allow conflict free drives and perceptions.

The interface between ARS and the body is called brain-socket. The brain-socket makes the ARS software independent from an agent platform which hosts the ARS decision unit. Different agent platforms can be attached to the ARS decision unit.

In the following the psychoanalytic ARS framework is used to be hosted by the NAO-robot-platform.

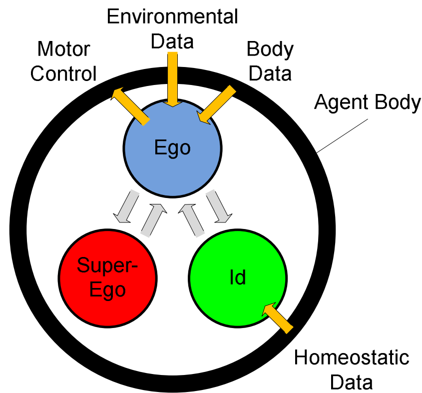


Fig. 2. Psychoanalytically inspired architecture

4 Implementation

The ARS project [4] implements psychoanalytic notions in artificial intelligence. In order to demonstrate the functionalities of ARS a multi-agent simulation environment was developed. This virtual world is called ARSini World. An agent is set into a virtual environment where the agent must process different tasks like finding nutrition, avoiding obstacles, and excrete.

The difficulty for the agent are conflicting situations where the satisfaction of bodily needs of the agent is impeded by enemy agents or dangerous situations. The task for the agent is to resolve these situations by using psychoanalytic tools and/or psychoanalytic defense mechanisms.

The final step is to transfer and couple the ARS decision unit to a NAO robot. This way, theoretical findings from the virtual simulation environment can be proved by using NAO in a real world application. This final step of coupling the ARS decision unit to NAO robot is further described in the current article in the next section.

5 Results

The aim of this article is to demonstrate the implementation of the psychoanalytic ARS control unit in the humanoid robot NAO. This way, theoretical findings of applying psychoanalysis in artificial intelligence can be demonstrated in a real world application.

To couple the ARS decision unit and NAO robot the brain-socket interface was developed and adapted to fit NAOs sensor- and actuator interface. The Webots simulation environment [21] helps adapting ARS interfaces to fit the NAOqi API.

Webots is used to test functionalities of NAOs interface before actually building ARS into NAO. NAOqi simulates the exact functionalities of NAOs actuators and sensors. Functionalities which work with NAOqi can be applied one to one to the real NAO robot. That, we can approve.

The following scenario is set up in the ARSini World to test the functionalities of the decision unit: The agent's task is to find and consume energy sources like cake or plant. Therefore, the agent can develop action plans with help of the decision unit. To form action plans the backward chaining approach [4, p. 99] is used. Action plans are generated by using a pre-defined rule. The action plans yield motory commands which control the actuators of the agent.

The same scenario as in ARSini World is now implemented in the Webots simulation environment to test the deployment with NAO. For NAO we needed to adapt the agent's perception capabilities and perception algorithms. NAO's perception engine is able to distinguish between different marker objects by size and shape. So we mapped the objects from ARSini World like cake and plant to marker objects for NAO's perception engine.

Fig. 3 shows these marker objects. We mapped the stone from the ARSini World to the TU-Wien symbol, the energy sources from ARSini World are represented by the number symbols 1 through 4, and ARSini World agents are mapped by using the number symbols 5 through 9.

By mapping the ARSini World symbols to NAO's marker symbols we were able to keep the architecture of ARS which compares perceived symbols with templates in a knowledge base.

Additionally to perception, we implemented three action commands for NAO's actuators: move forward, turn, and consume.

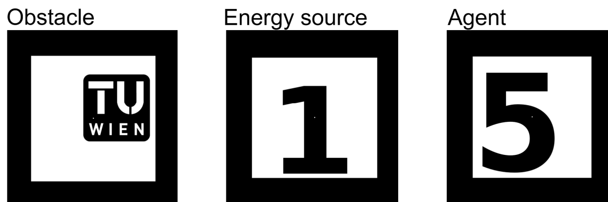


Fig. 3. Interpretation of marker symbols

6 Conclusion

This article reports the first ever implementation of a psychoanalytically inspired decision unit as control system for a biped humanoid robot. Previously, the developed architecture was evaluated using an artificial life simulation only. While this was a necessary intermediate step, we showed how to adapt the model and its database to an embodied real world agent, the NAO robot. The experiments show that the resulting behavior is comparable to the one observed in the simulation. Thus, the model is robust regarding calculation time differences (deliberation takes zero time in the

simulation) and different locomotion types. In the simulator a two wheeled simple locomotion is used, which is much simpler than the biped holonic drive of the robot.

References

1. Hsu, F.-H.: Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. Princeton University Press (2002)
2. Dietrich, D.: Evolution potentials for fieldbus systems. In: Proceedings of WFCS, pp. 145–146 (2000) (invited Talk)
3. Brainin, E., Dietrich, D., Kastner, W., Palensky, P., Rösener, C.: Neuro-bionic architecture of automation systems: Obstacles and challenges. In: Proceedings of 2004 IEEE AFRICON, vol. 2, pp. 1219–1222 (2004)
4. Zeilinger, H.: Bionically inspired information representation for embodied software agents. Ph.D. dissertation, Vienna University of Technology (2010)
5. Lang, R.: A decision unit for autonomous agents based on the theory of psychoanalysis. Ph.D. dissertation, Vienna University of Technology (2010)
6. Dietrich, D., Fodor, G., Zucker, G., Bruckner, D.: Simulating the Mind - A Technical Neuropsychoanalytical Approach. Springer, Wien (2009)
7. Soucek, C., Russ, G., Fuertes, C.T.: The smart kitchen project - an application on fieldbus technology to domotics. In: Proceedings of IWNA, p. 1 (2000)
8. Deutsch, T., Zeilinger, H., Lang, R.: Simulation results for the ars-pa model. In: Proc. 5th IEEE INDIN, June 23–27, vol. 2, pp. 995–1000 (2007)
9. Deutsch, T., Lang, R., Pratl, G., Brainin, E., Teicher, S.: Applying psychoanalytic and neuro-scientific models to automation. In: Proc. 2nd IET IE, vol. 1, pp. 111–118 (2006)
10. Novak, G., Mahlknecht, S.: Tinyphoon - a tiny autnomous agent. In: IEEE ISIE 2005, vol. 4, pp. 1533–1538 (2005)
11. Pfeifer, R., Scheier, C.: Understanding Intelligence. MIT Press (1999)
12. Deutsch, T., Muchitsch, C., Zeilinger, H., Bader, M., Vincze, M., Lang, R.: Cognitive Decision Unit Applied to Autonomous Biped Robot NAO. In: IEEE INDIN (2011)
13. Contact-Reactive Grasping of Objects with Partial Shape Information, 10 (2010)
14. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Open Source Software (2009)
15. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE J. Robotics and Automation, 14–23 (1986)
16. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press (1987)
17. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial Intelligence 33, 1–64 (1987)
18. Anderson, J.R., Lebiere, C.: The Atomic Components of Thought. Lawrence Erlbaum Associates (1998)
19. Franklin, S., Patterson, F.G.: The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. In: Proceedings of IDPT (2006)
20. Goertzel, B.: Patterns, hypergraphs and embodied general intelligence. In: Proceedings of the IJCNN, pp. 451–458 (2006)
21. Michel, O.: Cyberbotics ltd - webotstm: Professional mobile robot simulation. International Journal of Advanced Robotic Systems 1(1), 39–42 (2004)

Building iRIS: A Robotic Immune System

Dietmar Schreiner

Vienna University of Technology
Institute of Computer Languages, Compilers and Languages Group
Vienna, Austria
schreiner@complang.tuwien.ac.at

Abstract. Progress in robotics has led to complex autonomous and even collaborating robotic systems, fulfilling mission critical tasks in safety critical environments. An increase in capabilities and thus complexity consequently led to a dramatic increase in possible faults that might manifest in errors. Even worse, by applying robots with emerging behavior in non-deterministic real-world environments, faults may be introduced from external sources. Consequently, fault testing has become increasingly difficult. Both, software and hardware may fail or even break, and hence may cause a mission failure, heavy damage, or even severe injuries and loss of life. The ability of a robotic system to function in presence of such faults, so to become fault tolerant, is a continuously growing area of research. Our work meets this challenge by developing a mechanism for robotic systems that is capable of detecting defects, selecting feasible counter measures, and hence keeping robots in a sane and and consequently safe state. Inspired by biology, we conceptually aim at an immune system for a robot (RIS), which is able to detect anomalies, and which is able to autonomously counter them by appropriate means. This position paper outlines the requirements and research scopes that have been identified as relevant for the development of a robotic immune system.

1 Introduction

Robustness and dependability are key factors for today's as much as upcoming embedded computer systems not only in safety and mission critical domains but also in everyday life. Driven by progress and market the number of devices that operate within the real world and hence may harm people and environment, like autonomous vehicles, computer guided aircrafts, or reconnaissance and rescue robots, is constantly growing. At the same time, the complexity of those devices and applications–devices typically consist of a large number of networked processors, sensors and actuators–has reached a noticeable extent. Complicated tasks have to be fulfilled within a nondeterministic environment under strict consideration of safety guarantees, but also under harsh cost and energy constraints. Features like long-term autonomy and system adaptability have become an additional burden on robotic systems' software. Traditional techniques in software development that typically handle a precisely enumerated set of faults do not

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 150–155, 2012.
© Springer-Verlag Berlin Heidelberg 2012

scale well with the aforementioned increasing system complexity and the real world's nondeterminism.

We think that a nature inspired approach similar to the concept of a biological immune system could be able to provide a sound and robust solution for this robotic issue. Assuming that there exists no complete enumeration of faults that should be detected and handled over the lifespan of a system within an indeterministic environment, classical error handling techniques quickly reach their limits in terms of development cost and technical capability. An immune systems can overcome the problem of unknown and unexpected faults by detecting anything that is not a sane system state, and by utilizing adaptable rules on how to (re)establish a sane system. This concept was developed by evolution over millions of years, and is a promising approach for robust autonomous dependable systems within nondeterministic environments.

2 Scopes of Research for a RIS

To achieve the goal of nature inspired fault-tolerant self-healing software research has to be conducted at the intersection of five major scopes like depicted in Figure 1. Affordable devices that fulfill all necessary dependability constraints like safety or robustness [1] can only be built by joining results from all denoted domains: (i) Model Driven Development as a basis for code generation as much as for system verification, (ii) Static Analysis as a methodology to derive system properties from program codes as much as to guide compilation and code generation tasks, (iii) Programming Languages to provide proper means of abstraction and semantically enriched specifications, (iv) Reflective Computing as a methodology for run-time monitoring and supervision, and (v) Nature Inspired Computing as promising way to handle the systems' increasing complexity.

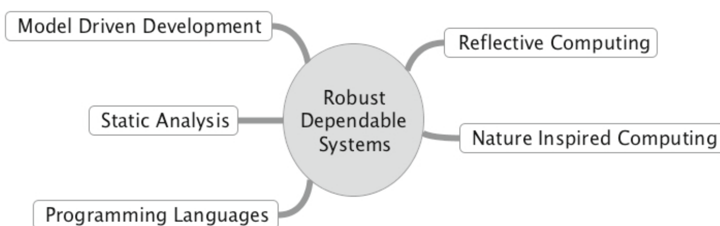


Fig. 1. Research Scopes

2.1 Model Driven Development

The complex nature of robotic applications—aspects like real time constraints as much as issues of concurrency or resource consumption have to be considered at the same time—introduces the need for a sound development methodology that not only provides separation of concerns but also allows the incorporation

of domain specific knowledge typically held by the robots' user rather than by programmers.

In model driven development views on the system that is built, the models, are specified at various levels of abstraction. These models are subject to so called model transformations, processes that automatically generate new artifacts like system configurations, new models, or even executable program code. Considering the requirements on robotic software engineering model driven development is a well suited methodology: specific aspects of the system under development are described in distinct views at proper levels of abstraction; domain specific know-how can easily be formalized by non-programmers as yet another view that can be transformed into technical views via automatic model transformations; system as much as application properties can be extracted from models via model transformations.

Our ongoing research within the domain of model driven development deals with automatic generation of synchronization artifacts for concurrent embedded systems applications [2]. Expected outcome is a sound methodology for automatic code injection and code generation for software sensors and reflective components in order to get run-time monitoring support for complex embedded systems as the backbone of an innate immune system.

2.2 Static Analysis

By examining executable code or even the executables' source code, many vital properties of a software system can be calculated before execution. These information can be used to automatically refactor and optimize code, detect various defects, or simply determine sane bounds for a program's execution.

The work relevant for a robotic immune system mainly aims at analyses that identify run-time properties before execution, like worst case loop-bounds, feasible paths, and WCET bounds, as a basis for an immune system's innate knowledge base, as much as at the identification of mount points for software sensors (which will be used later on in model driven injection (see Section 2.1)).

2.3 Programming Languages

Programming languages are the computer systems' interface for software developers. However, most programming languages used for embedded systems today are well suited for process centric or even hardware related software development, but do not meet upcoming requirements like distributed computing, concurrency, or dependability constraints. Additionally, those languages require a high expertise in programming and firm knowledge of the underlying hardware, which typically excludes experts from the devices' application domain from application development.

Visual programming languages, but also visual modeling techniques–we do consider a program's source code to be a model of the program–are one way to provide abstraction where necessary, and to lower the barrier for application domain experts to create or at least modify their own application. Additionally,

complex system constraints can be expressed in a comprehensible way, and can furthermore be used by transformation and compilation techniques to generate subsystems of an embedded immune system. The idea of graphical modeling is well accepted, its application to the automotive domain is for example outlined in [3] where UML 2.0 profiles were used to graphically specify a system's communication requirements.

2.4 Reflective Computing

Reflective computing denotes techniques that provide self-inspection of running systems. Although this idea is not new to computer science, it is part of the research agenda for a RIS. Introspection is an inevitable feature for any immune system, biological as much as artificial. Aiming at the automatic generation of an artificial immune system for a robotic device, software sensors have to be automatically deployed at compile time in order to disencumber application developers.

Research in that domain will hence deal with questions on identifying the proper locations for software sensors (this question goes hand in hand with static analysis) as much as a robust and resource constrained embodiment of those sensors.

2.5 Nature Inspired Computing

Nature has developed remarkable mechanisms for extremely complex tasks by evolution over a very long time. Consequently, those concepts can provide the solution to many open scientific but also engineering issues. However, the concepts have to be understood and simplified, abstracted in terms of a computer scientist, and adapted to specific needs.

The vision outlined in this paper is that of an artificial immune system for autonomous robust embedded devices. This system is inspired by biological immune systems, at least as far as they are understood today, and is custom tailored to the upcoming technical domain of collaborating robotic systems.

3 iRIS: An Overview

In terms of computer science, a biological immune system is a robust, multilayered, distributed system that is able to identify numerous pathogens. Its main responsibility is to counter act harmful effects to keep the organism in a sane state. An artificial immune system is meant to do exactly the same for computers and in our case robotic systems. Harmful effects can be detected and can be overcome autonomously.

In [4] one bio-inspired algorithm, the negative selection algorithm, is described, which is based on the detection of self from non-self. Dedicated immune cell types like lymphocytes have receptors that allow them to bind to specific

proteins. During maturation these cells are 'trained' on proteins that are naturally present in the organism, the self-antigenes. Lymphocytes which spuriously bind to self-antigenes are destroyed immediately (apoptosis). After reaching maturity, the trained cells are spread over the organism. If they bind to a protein now, this is a clear indication of a non-self protein, a pathogen.

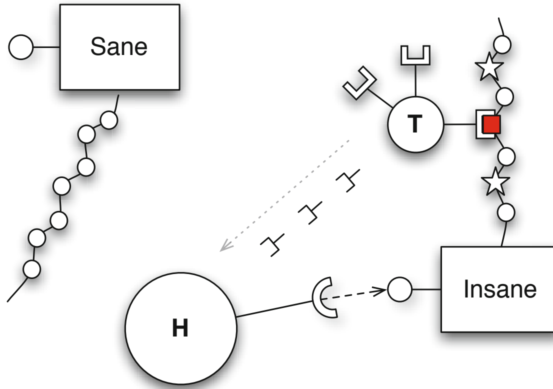


Fig. 2. Immune reaction to faulty component

iRIS (innate Robotic Immune System) is structured in accordance to a biological immune system: Within the organism (the robot), distributed autonomous light-weight processes, so called T-processes, constantly monitor the system's sanity via their receptors, e.g., software-sensors, execution monitors, or even hardware sensors. Antigenes are sequential representations of specific states within the robot, and may be defined and classified at development but also dynamically at run-time. Detection of non-self antigenes is achieved in two ways: (i) T-processes undergo a process of maturation, which means they are trained to detect self-antigenes on a running system. (ii) T-processes utilize 'genetic knowledge', represented by predefined rules and parameters. On detection of a non-self antigene, T-processes activate H-processes, which are in charge of eliminating anomalies (e.g., by restarting malfunctioning components of a robot, reinitializing affected structures, or recalibrating sensors).

Figure 2 depicts the conceptual idea behind an iRIS immune reaction: It shows two components, *Sane* and *Insane*, both exposing their associated antigene. As *Insane* is malfunctioning, the T-process T detects this non-self behavior and notifies specialized H-processes. One H-process finally docs at *Insane's* maintenance interface to counter-act the fault. In addition, our observations show that typical faults within robotic systems require complex repair actions at distributed subsystems. For that reason, H-processes may emit messenger antigenes to trigger additional repair activities at all affected subsystems.

Compared to existing work, iRIS covers aspects of autonomic computing like summarized in [5]. It does reflective computation [6,7] at run-time in a bio-inspired way, using knowledge extracted from static analyses and system models

at development and compile time. A model driven development methodology as much as static analyses come to use to extract the 'genetic knowledge' of iRIS. In addition, iRIS incorporates dynamic means of machine learning, also inspired by natural immune systems, to cope with unforeseen faults.

4 Conclusion

This paper has outlined the basic idea for an innate robotic immune system (iRIS), which aims at robust dependable robots. In order to build such a system research has to be conducted at least in five domains within computer science: (i) Model Driven Development, (ii) Static Analysis, (iii) Programming Languages, (iv) Reflective Computing, and (v) Nature Inspired Computing. Merging results from these domains will enable the development of an innate immune system capable of self/non-self discrimination, which denotes the baseline for further adaptable systems.

References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
2. Schreiner, D., Puntigam, F.: Robots, Software, Mayhem? Towards a Design Methodology for Robotic Software Systems. In: Supplemental Volume of the Eight European Dependable Computing Conference (EDCC 2010), pp. 31–32 (April 2010) ISBN: 978-84-692-9571-7
3. Schreiner, D., Göschka, K.M.: Modeling Component Based Embedded Systems Applications with Explicit Connectors in UML 2.0. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC 2007), pp. 1494–1495. ACM Press, New York (2007)
4. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy (1994)
5. Dobson, S., Sterritt, R., Nixon, P., Hinchey, M.: Fulfilling the vision of autonomic computing. Computer 43(1), 35–41 (2010)
6. Maes, P.: Concepts and experiments in computational reflection. In: OOPSLA, pp. 147–155 (1987)
7. Rodríguez, M., Fabre, J.-C., Arlat, J.: Wrapping Real-Time Systems from Temporal Logic Specifications. In: Bondavalli, A., Thévenod-Fosse, P. (eds.) EDCC 2002. LNCS, vol. 2485, pp. 253–270. Springer, Heidelberg (2002)

Towards Reorientation with a Humanoid Robot

Dietmar Bruckner[1], Markus Vincze[2], and Isabella Hinterleitner[1]

[1] Institute of Computer Technology, Vienna University of Technology,
Gusshausstrasse 27/384, 1040 Vienna, Austria, Europe
{bruckner,hinterleitner}@ict.tuwien.ac.at

[2] Automation and Control Institute, Vienna University of Technology,
Gusshausstrasse 29/384, 1040 Vienna, Austria, Europe
vincze@acin.tuwien.ac.at

Abstract. Current cognitive vision systems and autonomous robots are not able to flexibly adapt to novel scenes. For example, when entering a new kitchen it is relatively simple for humans to adapt to the new situation. However, there exist no methods such that a robot holds a generic kitchen model that is then adapted to the new situation. We tackle this by developing a hierarchical ontology system linking process, object, and abstract world knowledge via connecting ontologies, all defined in a formal description language.

The items and objects and their affordances in the object ontology are either learned from 3D models of the Web or from samples. We bind the features of the learned models to the concepts represented in the ontology. This enables us to actively search for objects to be expected to be seen in a kitchen scenario. The search for the objects will use the selection of cues appropriate to the relevant object. We plan to evaluate this model in three different kitchens with a mobile robot with an arm and further with Romeo, a humanoid robot designed by Aldebaran to operate in homes.

Keywords: Cognitive Robotics, Situated Vision, Ontology, Reorientation.

1 Introduction

Our goal is to understand the necessary preconditions for perception and action in a context which is familiar in an *abstract* sense (e.g., tea making), yet not in a given *concrete* setting (a kitchen not seen before). Our vision is to give an explanation of this interplay between perception and vision and to provide a representation of these different types of knowledge, such as different ontologies [1] (encompassing task, process, affordance [2] and object knowledge) that are used at different stages. This abstract knowledge needs to be modeled such that it can be tethered [3] to actual objects and the affordances provided by a new situation. We see several major advantages of this approach:

- Clean separation of top-down abstract knowledge and bottom-up concrete data.
- Insights of what an abstract database needs to provide in order to guide visual input towards the task affordances and processes (the actual robot grasping and manipulation).

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 156–161, 2012.
© Springer-Verlag Berlin Heidelberg 2012

- A thorough understanding of what vision needs to provide to perceive these affordances given a specific task and robot embodiment, we refer to this as the *situatedness* of vision [4].
- The natural inclusion of *attention-based* vision to re-orientate in novel environments extending present artificial saliency maps to situated robots.
- And, formal means to study how to flexibly adapt to novel settings given the genericness of the ontological knowledge.

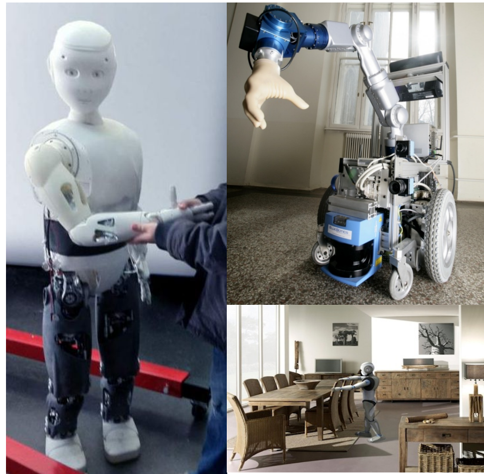


Fig. 1. Robots to show the situated vision approach: James with 7DOF arm and hand prosthesis (top right) and Romeo by Aldebaran (140cm tall, right bottom) shaking hands with a boy (left)

2 Motivation

In his seminal work Michael Land investigated human eye movement when performing a tea-making task in a kitchen at their university [5]. The important aspect is that the subjects had not seen the kitchen before, but before recording the sequence it took them about 30-40 seconds to re-orient in the new setting. During the recorded eye fixations while making tea all fixations are “relevant” to the task, clearly indicating that the human has built up a full mental representation of the kitchen setting. The fixations fall into four action types: locating (objects), directing (an object towards a new location), guiding (the interaction of one object with another), or checking (the state of a variable). The authors outline different levels of description, starting from a higher-level goal (“make the tea”) to single steps (such as “transport to sink”).

It is striking that this work focuses on questions involved in the “execution” of the task and not the crucial re-orientation phase before the task can be executed so smoothly. We are, however, interested in the seconds *before* the test subject starts making the tea, i.e., the seconds that involve the “binding” of needed parts of the task to the concrete objects found in the current situation [6].

3 State of the Art

3.1 Situated Vision

AI's notion of "situatedness" [7, 4] emphasizes the necessity to focus the operation of an agent enabling it to operate reliably in an uncertain, changing environment. The rationale is that domain knowledge scales the possibilities found in such a setting.

Accomplishing tasks in an everyday setting, especially in a home environment, is a great challenge for robots and vision alike. The reason is the inherent uncertainty that is imposed by the lack of controlling the environmental conditions. Humanoids or rolling torsos can approach an immediate target but environments are fully modeled and objects need to be clearly colored or textured [8, 9]. The environment can be reconstructed from tracking features using a Manhattan assumption [10], however obtaining high level structure requires constraints such as given models [11] or recurrent items [12]. First work shows how to systematically search for objects using bottom up information in a room with tables and sofas [13], a few shelves with a humanoid robot [14], or learning views related to a bicycle [15]. The most cognitive approach learns and reasons about object to room relations [16], which is a first step towards reorienting in a new setting.

3.2 Formalizing Task Knowledge

Task planning is a key issue in robotics. [17] defines a task matrix and focuses on developing a set of complex behaviors using manually-devised connections between task programs. [18] studies industrial automation tasks and distributes tasks amongst agents, as robots in a flexible factory production have to solve industrial automation tasks. The complete ontology of how to achieve a task is split up in the sum of partial task descriptions in the agents. A theoretical approach for an ontology of robot tasks is presented in [19].

While psychology is debating the necessity and existence of explicitly accessible representations [20] we have to find ways to assign tasks to robots to make them interact with the environment. The robot has to check the environment for available resources. In our situated vision approach this is achieved via features or objects in the environment that have to be recognized in terms of *affordances* [2]. Gibson defines an affordance as follows:
Affordances relate the utility of things, events, and places to the needs of animals and their actions in fulfilling them [...] Affordances themselves are perceived and, in fact, are the essence of what we perceive.

3.3 Visual Attention

The concept of visual attention is well investigated in human vision. Many psychological models of visual attention exist (cf. overviews in [21, 22]). Among the best known models are the Feature Integration Theory [23] and the Guided Search model [24]. All models have in common that several feature channels are processed in

parallel and fused to a single saliency map. This map highlights the most interesting regions in a scene and guides the focus of attention. There are controversies about which features guide the deployment of attention. Some cues however are undoubted to belong to these basic features: color, motion, orientation and size. Other studies also support depth cues, shape and curvature [25].

4 ENTER Approach

We introduce a concept of four ontologies, hierarchically ranging from high level tasks to low level objects. Although the planned nested ontology has to fulfill a high level task, at the level of the task ontology only the object class is defined.
In the first step a task is broken into subtasks (processes). Objects have to be identified to meet requirements of a process (affordances). Thus, on the lower level of ontologies we have a basic object ontology and the affordance ontology.
The fact that only the class is defined at task level makes the whole ontology very much light weighted and flexible.

The robot evaluates the environment according to affordances of, e.g., water supply, making hot water, containers for tea bags and the like. To do so the robot has to identify the function of an object (where the term “object” also includes, e.g., flat surfaces that can be used to place a cup or saucer). Affordances are more than descriptions of objective functionality. Rather than describing the function of an object they describe what impact an object has on the entity that interacts with the object. Thus, it depends not only on the object, but also on the agent and on what the environment can offer. Depending on the shape and degrees of freedom of a robot hand an object may be lift-able or even carry-able or it may not be movable at all and thus considered as an obstacle. Depending on the size of the robot hand an object may be considered as graspable or not. Below object level an affordance-based ontology is constructed: it links objects to robot actions that could be performed to complete a task given the robot's capabilities.

The task ontology is considered as a domain ontology and on the highest level it contains the processes required to fulfill a task including its context.

Finally, the object ontology contains all objects that are needed for a task, such as cup, kettle, milk, fridge, tap, etc. The object ontology subsumes the objects’ affordances representing the semantic knowledge.

The example in Fig.2 shows how the FillCup scenario can be fulfilled based on the model of hierarchical information processing. First, a camera gives information, whether the kettle and/or the cup are already on the table. Then, giving direct feedback during execution of the process, sensors, such as the contact sensors, detect, whether the robot is grasping the kettle correctly. Finally, the sensors observe, if the arm is manipulated in order to pour the water from the kettle in the cup. The example for the robot pouring the water works given the fact that the cup is already in the range of the machine. If this is not the case, the world knowledge will be consulted in order to search where a cup could be located. As is stated there, the robot would then start searching in cabinets for a cup.

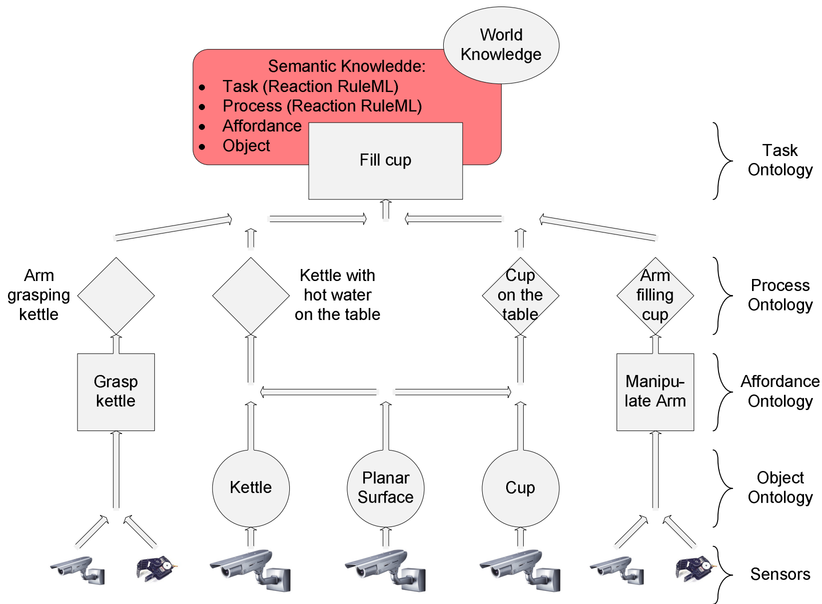


Fig. 2. ENTER Ontologies for the concrete example of filling the tea cup with hot water from the kettle. On the right the different ontologies are listed while on the left the dependencies between instances of the ontologies are shown.

The novelty is that this neuro-symbolic network model can be described by using rule systems, where each network node becomes a rule, representing the node inputs as premises and the node outputs as conclusions. This enables a high-level specification of neuro-symbolic networks and the use of the RuleML format for network interchange [26].

References

1. Shadbolt, N., Hall, W., Berners-Lee, T.: The semantic web revisited. IEEE Intelligent Systems 21, 96–101 (2006)
2. Gibson, J.J.: The ecological approach to visual perception. Houghton Mifflin, USA (1979)
3. Sloman, A.: Getting meaning off the ground: Symbol-grounding vs symbol-tethering (March 2002), `http://www.cs.bham.ac.uk/research/projects/cogaff/talks/` (April 2009)
4. Pfeifer, R., Scheier, C.: Understanding Intelligence. MIT Press, Boston (2001)
5. Land, M., Mennie, N., Rusted, J.: The roles of vision and eye movements in the control of activities of daily living. Perception 28(11), 1311–1328 (1999)
6. Tatler, B.W., Hayhoe, M., Land, M.F., Ballard, D.H.: Eye guidance in natural vision: Reinterpreting salience. Journal of Vision 11(5), 1–23 (2011)
7. Clancy, W.J.: Situated Cognition. Cambridge University Press, Cambridge (1997)
8. Gravot, F., Haneda, A., Okada, K., Inaba, M.: Cooking for humanoid robot, a task that needs symbolic and geometric reasoning. In: IEEE International Conference on Robotics and Automation, ICRA 2006, pp. 462–467 (May 2006)

9. Tenorth, M., Klank, U., Pangercic, D., Beetz, M.: Web-enabled robots. IEEE Robotics Automation Magazine 18(2), 58–68 (2011)
10. Flint, A., Mei, C., Murray, D., Reid, I.: A Dynamic Programming Approach to Reconstructing Building Interiors. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part V. LNCS, vol. 6315, pp. 394–407. Springer, Heidelberg (2010)
11. Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M.E., Beetz, M.: Functional object mapping of kitchen environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008, pp. 3525–3532 (September 2008)
12. Ruhnke, M., Steder, B., Grisetti, G., Burgard, W.: Unsupervised learning of compact 3d models based on the detection of recurrent structures. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2137–2142 (October 2010)
13. Meger, D., Forssn, P.-E., Lai, K., Helmer, S., McCann, S., Southey, T., Baumann, M., Little, J.J., Lowe, D.G.: Curious george: An attentive semantic robot. Robotics and Autonomous Systems 56(6), 503–511 (2008); from Sensors to Human Spatial Concepts
14. Andreopoulos, A., Hasler, S., Wersing, H., Janssen, H., Tsotsos, J.K., Korner, E.: Active 3d object localization using a humanoid robot. IEEE Trans. on Robotics 27(1), 47–64 (2011)
15. Meger, D., Gupta, A., Little, J.J.: Viewpoint detection models for sequential embodied object category recognition. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 5055–5061 (May 2010)
16. Wyatt, J.L., Aydemir, A., Brenner, M., Hanheide, M., Hawes, N., Jensfelt, P., Kristan, M., Kruijff, G.M., Lison, P., Pronobis, A., Sjöö, K., Vrecko, A., Zender, H., Zillich, M., Skocaj, D.: Self-understanding and self-extension: A systems and representational approach. IEEE Transactions on Autonomous Mental Development 2(4), 282–303 (2010)
17. Drumwright, E.: The task matrix: An extensible framework for creating versatile humanoid robots. In: IEEE Intl. Conf. on Robotics and Automation, ICRA (2006)
18. Feng, Q., Bratukhin, A., Treytl, A., Sauter, T.: A exible multi-agent system architecture for plant automation. In: 5th IEEE International Conference on Industrial Informatics, pp. 1047–1052 (2007)
19. Hidayat, S.S., Kim, B.K., Ohba, K.: Learning affordance for semantic robots using ontology approach. In: IEEE/RSJ International Conference on Intelligent Robots and Systems IROS, pp. 2630–2636 (2008)
20. Haselager, P., de Groot, A., van Rappard, H.: Representationalism vs. antirepresentationalism: a debate for the sake of appearance. Philosophical Psychology 16, 5–23 (2003)
21. Bundesen, C., Habekost, T.: Attention. In: Lamberts, K., Goldstone, R. (eds.) Handbook of Cognition, Sage Publications, London (2005)
22. Frintrop, S., Rome, E., Christensen, H.I.: Computational visual attention systems and their cognitive foundation: A survey. ACM Trans. Applied Perception 7(1), 1–46 (2010)
23. Treisman, A.M., Gelade, G.: A feature integration theory of attention. Cognitive Psychology 12, 97–136 (1980)
24. Wolfe, J.M.: Visual search. In: Pashler, H. (ed.) Attention, pp. 13–74. Psychology Press, Hove (1998)
25. Wolfe, J.M., Horowitz, T.S.: What attributes guide the deployment of visual attention and how do they do it? Nature Reviews Neuroscience 5, 1–7 (2004)
26. Boley, H.: Posl: An integrated positional-slotted language for semantic web knowledge, ruleml draft (2004), `http://www.ruleml.org/submission/ruleml-shortation.html`

Monitoring Anomalies in IT-Landscapes Using Clustering Techniques and Complex Event Processing*

Matthias Gander, Michael Felderer, Basel Katt, and Ruth Breu

Institute of Computer Science, University of Innsbruck, Austria
{matthias.gander,michael.felderer,basel.katt,ruth.breu}@uibk.ac.at

Abstract. Monitoring the behavior of IT-landscapes is the basis for the detection of breaches of non-functional requirements like security. Established methods, such as signature-based monitoring extract features from data instances and compare them to features of the signature database. However, signature-based monitoring techniques have an intrinsic limitation concerning unseen instances of aberrations (or attacks) because new instances have features which are not yet recognized in the signature database. Therefore, anomaly detection has been introduced to automatically detect non-conforming patterns in data. Unfortunately, it is often prohibitively hard to attain labeled training data to employ supervised-learning based approaches. Hence, the application of non-supervised techniques such as clustering became popular. In this paper, we apply complex event processing rules and clustering techniques leveraging models of an IT-landscape considering workflows, services, and the network infrastructure to detect abnormal behavior. The service and infrastructure layer both have events on their own. Sequences of service events are well-defined, represent a workflow and are counter-checked via complex event processing rules. These service events however trigger infrastructure events, like database activity, and network traffic, which are not modeled. These infrastructure events are then related to the appropriate call traces and clustered among network profiles and database profiles. Outlying service events, nodes, and workflows are detected based on measured deviations to clusters. We present the main properties of our clustering-based anomaly detection approach and relate it to other techniques.

1 Introduction

In high security industries, especially in multi-user collaborative information systems (CIS), it is common to specify security requirements, in more or less formal models. For CIS in the health-care domain for instance, the *integrating the health-care environment* (IHE) standards committee[1] provides requirement documents

* This work is supported by QE LaB - Living Models for Open Systems (FFG 822740), COSEMA - funded by the Tiroler Zukunftsstiftung, and SECTISSIMO (P-20388) FWF project.

[1] http://www.ihe.net/, Accessed: July 20, 2012.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 162–180, 2012.
© Springer-Verlag Berlin Heidelberg 2012

for authentication, authorization, and auditing. To make sure that these security requirements are met correctly, different monitoring solutions have been established, e.g. *intrusion prevention/detection systems* (IPS/IDS). The *Common Intrusion Detection Framework* (CIDF), a working group founded by DARPA in 1998, and later changed to *Intrusion Detection Working Group* (IDWG) defined a common architecture for IDS by using four functional components [1].

– E blocks, also called Event-boxes, are basically configured *sensors* that monitor given target systems.
– D blocks, also called Database-boxes, store event data received by the E blocks and allow further processing.
– A blocks, also called Analysis-boxes, are components that allow further analysis on data within D blocks.
– R blocks, also called Response-boxes, allow to react on alerts, e.g. stop execution of a compromised system.

Differences within the blocks imply a variety of detection systems [1]. If the E blocks explicitly collect network traffic, and are, therefore, stationed within routers/switches, we speak of *network intrusion detection systems* (NIDS). If, on the other side, sensors are stationed within hosts to monitor, for instance, log files or application behavior, we speak of host-based intrusion detection systems (HIDS). A very common technique for A blocks involves signature-based methods. Features, extracted from event data in D blocks are compared to features in attack signatures that are provided by experts. Other approaches, e.g. anomaly detection, often make use of machine learning-based algorithms. Both approaches, signature-based and anomaly-based, have strengths and weaknesses.

Signature-Based Approaches. If a signature-based approach finds an attack, it is very probable that it is indeed a true-positive. Unfortunately signature-based methods have the inherent limitation that they always have to consult the signature database to match detected features with the information therein. If a new attack is out, it is probable that the signature database does not contain the latest attack pattern.

Anomaly-Based Approaches. Anomaly-based detection techniques compared to signature-based approaches are known to detect previously unseen patterns, i.e. determine if an entity (user, service, data flow etc.) is sufficiently different from the average behavior. Anomaly-based techniques model "normal" behavior of the system that needs protection, and generate an anomaly alert whenever the deviation between features of newly registered events and normal features exceed some threshold. We are interested in machine-learning based approaches, i.e. supervised and unsupervised learning. Supervised machine-learning has some major drawbacks, the first one being the dependency on labeled training instances, which is not always easy to procure. And the second is that training instances are susceptible to be trained by an attacker [1]. In this work we make use of the assumption that the behavior of the network is mostly normal ($\eta\%$) and the abnormal behavior is just a tiny fraction of the overall behavior ($1 - \eta\%$) [2].

Therefore it becomes possible to apply unsupervised-machine learning to detect this change in behavior. Unsupervised-learning can roughly be classified in, nearest-neighbour, rule-mining, statistical, and clustering techniques. Each of which have advantages and disadvantages, depending on how they are used, see Chandola et al. [3]. For our purpose, grouping anomalous instances, clustering seems best suited. The disadvantages of clustering, i.e. the complexity of clustering algorithms, a possibly missclassification, can be mitigated by leveraging optimized algorithms and assumptions [2].

Complex Event Processing (CEP). In a mid–to–upper size IT infrastructure (e.g., in the health-care sector) it is very common that many service events are generated in a short amount of time, almost isochronal. Composition of multiple services necessitates patterns to determine the correct execution order of services. This is the task of *Complex Event Processing* (CEP). Its purpose is to derive a more meaningful, a more complex, event from those which are collected. An example is a CEP system in a car which alerts the user of a flat tire. The combination of pressure loss in some tire and possibly a short period during which the pressure was lost could be the "simple" events from which the alert was generated [4]. Much research has been invested in query languages to handle the stream of events in query-based languages similar to SQL[2], ESPER[3], Oracle CEP[4], Coral8[5] and Aleri[6].

Contribution and Structure

In this paper we present a general anomaly detection framework for a multi-user CIS to support real-time outlier detection for insider threats based on the analysis of event streams. We provide an *IT-landscape* metamodel as vocabulary for solution architects to build the concrete landscape model, i.e. the workflows, the services, and the nodes they rely on, which represents the CIS. Using the information from the, thus, created model, the framework can provide stateful monitoring of workflows as well as anomaly detection. In these landscape models each workflow can be characterized as a sequence of CEP rules to describe distinct steps of a concrete workflow. Since the vocabulary provides means to attach service instances to workflow events, they can be related. During execution of the system, service events are mapped to infrastructure clustering-instances, that represent normal and abnormal behavior of the network, database, and users, respectively. Using the above mentioned CEP rules the system detects workflow attacks, and through clustering of infrastructure events we are able to create normality profiles, detect outliers and raise alerts accordingly. The contribution of our framework will be threefold:

[2] `http://www.w3schools.com/sql/default.asp`, Accessed: July 20, 2012.
[3] `http://esper.codehaus.org/`, Accessed: July 20, 2012.
[4] `http://tinyurl.com/OracleCEP`, Accessed: July 20, 2012.
[5] `http://tinyurl.com/Coral8CEP`, Accessed: July 20, 2012.
[6] `http://tinyurl.com/AleriStreaming`, Accessed: July 20, 2012.

1. A UML metamodel [5] to specify an IT-landscape for multi-user CIS which includes the network infrastructure, service events and workflows. As we will see, model information provides information to link events to their origin in the model (*traceability*), but also allows the creation of monitoring rules (CEP).
2. The detection of (a) workflow attacks via CEP rules by breaking down the workflow into distinct events, and (b) anomalous user and network activity by clustering event streams.

In Section 2 we present a case study, possible threats (threat model), assumption and objectives. This is followed by Section 3 which presents the framework, compliance detection. Section 4 and 5 discuss related and future work respectively.

2 Basic Context

In this section we discuss our motivating example, present our threat model, and provide details about concepts, such as the layered approach we use, and the events we are interested in.

2.1 Case Study

We elaborate our framework based on a running example taken from a workflow in the health-care domain. In this scenario a doctor (document consumer), logged in on a host, wants to retrieve a patient medical record (sensitive information) from a document repository over an XACML [6] infrastructure. By this example we discuss the behavior of a workflow, demonstrate possible attacks, and finally show how the proposed framework diffuses them.

The workflow makes use of SOAP web services (HTTP). For the description we make use of the standard BPMN and workflow notions, i.e. actors and tasks [7]. We distinguish among multiple actors in our example: The *Identity Provider*, which identifies nodes and users, but also assigns roles. The *Document Consumer*, which is the source of a patient document query, or document change request, for instance a host requesting a patient medical record (PMR). The *Document Repository*, which is the location of the patient data. Access control is offered through an XACML architecture.

A document retrieval transaction, which involves polling a patient's confidential information consists of two tasks, namely authentication, and document retrieval. Below we explain the tasks in the sample workflow from Figure 1:

- *Provide Credentials*: The document consumer (DC) forwards his credentials to the identity provider.
- *Authentication*: The identity provider queries a database interface service to validate the credentials and assigns a role to the user and answers with a SAML 2.0 assertion containing an authorization statement.
- *Retrieve Document Query*: After the document consumer receives this ticket, the actual retrieval can start. The consumer sends a retrieve document query together with his ticket to the document repository.

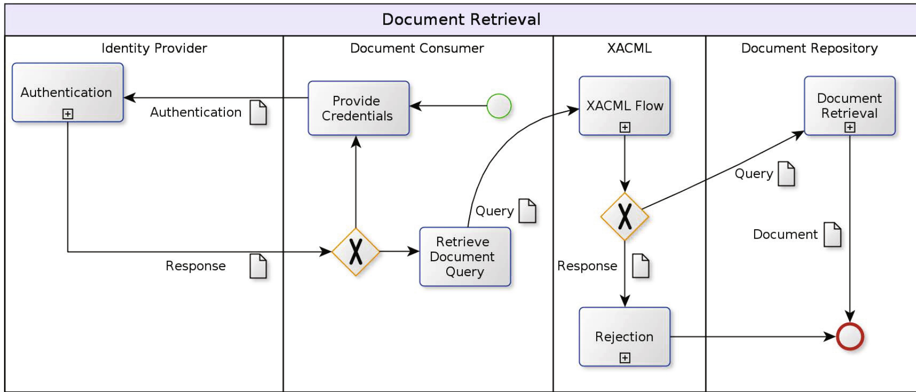


Fig. 1. An exemplary document retrieval workflow

– *XACML Flow*: The request is intercepted by the XACML infrastructure.
  1. The *policy enforcement point* (PEP) attached to the document repository calls the PIP to deduce attributes for the document that needs to be retrieved.
  2. The *policy information point* (PIP) returns the appropriate attributes.
  3. The PEP forwards the query and resources to the PDP and awaits a decision.
  4. The *policy decision point* (PDP) acquires policies for the target from the PAP.
  5. The *policy assertion point* (PAP) returns the appropriate policies and via rule combining algorithms derives a decision.
  6. The PEP receives the decision and either (depending on the ruling of the PDP) allows or rejects the access.
– Now either the rejection message (*Rejection*), due to unauthorized access, or the document itself (*Document Retrieval*) is transmitted to the document consumer, concluding the workflow.

It is possible to track the execution of such a workflow in granular fashion on the service level, i.e. by tracking some/all service events (calls). Taking into account that service events are related to infrastructure events, for instance, network events (TCP, UDP) and database access events (CRUD) it becomes possible to relate infrastructure traffic to service events.

2.2 Problem Statement and Assumptions

An attacker who wishes to gain confidential information is bound to steal credentials from an asset (doctor) or hacking into a node (host, router, ...) of the network. For the intruder, it is often prudent to install backdoors, i.e. by patching or replacing code [8, 9], to (i) gain easy access without the need to re-exploit a vulnerability and (ii) tweak the behavior of a system in ones favor. Attack

techniques are manifold [10, 11]. In our experience they manifest either in workflows, fiddling with the execution flow, or a layer beneath, tampering with the infrastructure.

Workflow Threats. A serious way to deal damage to a company is to deliberately sabotage the service invocation sequence. For instance if an attacker manages to bypass checks, i.e. for sufficient funds, in a shopping application. In our case study a likely attack target is the XACML infrastructure. Assume an attacker wishes to grant access for certain individuals to confidential data by changing the internal behavior of the XACML architecture. More technically speaking, assume, the PEP does not invoke the PDP in the access control flow and allows selected individuals access to PMRs, although they are usually not permitted. Another example is the communication between PDP and PIP. The PDP is supposed to retrieve document attributes from the PIP to base its decision making. What if the PDP does not? Services along the way of document queries are also susceptible to attacks, for instance, replay-attacks or fake queries (false tickets). Whatever the attack, it will manifest in missing events, wrong sequences of events, sometimes too many (replay attacks), along the execution of the workflow.

Infrastructure Threats. Our objective lies in network-based attacks that manifest as odd connection patterns meanwhile a service event takes place.

Example 1. In the following Table 1 we illustrate an anomaly that portraits an access rights manipulation in the activity *Retrieve Document Query.* The queries shown in $\mathrm{Ev}_{5,6}$ (from the document consumer with UID_x and requested resource object, PMR_{12}), result in *Authorization Denied* events emitted from the PDP. Immediately after the second *Authorization Denied* event, the database is

Table 1. Access Rights Manipulation

Event	Layer	Source	Dest	App	Ident	Object	Time	Type
Ev_1	Service	DC	DR	–	UID_x	PMR_{12}	7.22 pm	Query
Ev_2	Service	DC	DR	–	UID_x	PMR_{12}	7.22 pm	Retrieve
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Ev_3	Infrastructure	DC	–	Postgres SQL	UID_x	PMR_{12}	7.23 pm	Read
Ev_4	Infrastructure	DC	–	Postgres SQL	PDP	PMR_{12}	7.23 pm	Read
Ev_5	Service	PDP	PEP	–	UID_x	PMR_{12}	2.01 am	AuthDenied
Ev_6	Service	PDP	PEP	–	UID_x	PMR_{12}	2.02 am	AuthDenied
Ev_7	Infrastructure	DC	–	Postgres SQL	UID_x	PMR_{12}	2.02 am	Update

accessed at the document repository for the same resource and more importantly from the same user, shown in Ev_7. Above all, as we will show, the time of the query is very unusual for the user UID_x.

Example 2. After the *Document Retrieval* from the DR has been sent to the DC, the DC starts a TCP based communication to two hosts, C and D, in the network. C is used for backup services, yet D, in this instance, represents a malicious host that collects confidential data. A plethora of other potential threats exist in this perspective, starting from web service attacks,[7] possible buffer overflows in "low-level" services like, FTP, and badly configured access control, i.e. allowing SMB null sessions. Dealing with the latter attacks is beyond the scope of this paper and partially covered by existing research [1, 12, 13]. Our concern is to detect anomalous communication that is triggered by service calls and anomalous database activity in relation to service calls. Service calls themselves may produce anomalies, e.g. service calls at unusual hour, service calls from unusual nodes, but also a change in flow behavior of service calls (for instance payload anomalies).

2.3 Course of Action and Objectives

In our framework we consider several layers of abstraction through the use of an IT-landscape model. This allows us for instance to detect possible XACML flow anomalies. The idea of splitting an enterprise model into multiple layers is not new, we build on the ideas of Breu et al. [14, 15]. The top layer is the *workflow layer*, that contains sequences of activities. These activities determine the sequence of a workflow. The *service layer* consists of services in the network that execute activities. The bottom layer is the *infrastructure layer*, in here, nodes host services and represent the backbone of executing workflows.

Events we consider are extracted from the *infrastructure* and *service* layer. Infrastructure events consist of network and database events. Network events are UDP and TCP packets that are aggregated as communication flows among two nodes. These flows have multiple characteristics, such as: source, destination, ports, time, among others. For database applications (especially in the health-care domain) it is essential to know all the details of database access, i.e. everything about create, read, update, and delete operations (CRUD). For this purpose database events need to contain a user id, the object id being accessed, the time, the type of the CRUD operation and the originating host. Workflow events are omitted, we derive workflow information implicitly over service events. Service events are calls from one service to another containing a type. In this work we cover a set of standard types, *authentication, authorization, query, retrieve, delete, update, create* and *forward.*

3 Clustering-Based Anomaly Detection Framework

In this section we discuss our anomaly detection framework in detail. We show how the model information in the IT-landscape model can be used to extract valuable information to create CEP monitoring rules and how infrastructure events can be related to service calls to attain network and database profiles of service calls.

[7] `http://www.ws-attacks.org`, Accessed: July 20, 2012.

3.1 IT-Landscape Metamodel

Domain-specific languages (DSL) are a common way to provide vocabulary for experts to let them express their knowledge of the system via textual or graphical models. Textual as well as graphical DSLs are not uncommon in enterprise modelling, for instance [14–16]. These models can later be accessed for look-ups, reasoning, and/or code generation.

In our case, the metamodel is a DSL that allows the creation of a model that in turn allows harvesting information and monitoring rule-generation concerning multi-user CIS, i.e. multi-tier network infrastructures, comprised of workflows, services, and hosts. The model we created reuses concepts from Breu et al. [14, 15], for example the introduction of multiple conceptual layers. We follow an event-driven process chain paradigm [17]. This has the benefit that services can be represented via events and, thus, need not be modelled explicitly. A workflow activity, therefore, is not modeled via services and their call-sequence but rather as a series of events.

Our model contains three layers, *Workflow, Service* and *Infrastructure*. The workflow layer contains three classes, these are *WF Activity, Role*, and *Actor*. Workflow activities but also service events are connected via arcs (*Arc*) that are of different types *AND, OR, XOR, SEQ*. SEQ denotes that if said arc lies between two workflow activities A and B, then A is followed by B. *AND, OR* and *XOR* are ternary arcs that relate events in a fashion known from boolean operations. For instance A XOR B,C denotes that after A either B or C is executed. Each activity is attached to a role. A role is a set of responsibilities and obligations for a stakeholder. A set of actors is associated to roles which are uniquely identifiable, via an *Identifier*. Services are not modeled directly, but rather as *ServiceEvent* of various types (*EventTypeEnum*). Event emitters are services, on top of network nodes. Hence, among other features provided by the service event, i.e. variable ones like timestamps and session ids (to identify the *Actor*), we assume a source and a destination pointing to the nodes that took part in the event. This allows us to connect the service layer to the infrastructure layer. Nodes can be of various type (*NodeType*), this makes it easier to map events to their corresponding workflow activity during runtime.

Each one of the many association relations denotes the use of the element. *Identifier* defines the set of identifiers, i.e. all elements are connected to it via *identifiedBy*, such as nodes, service events, and actors are identified by it (via UUID and a location). Workflow activities are executed via services, the execution order is in form of service events. The real elements doing the execution are nodes from the infrastructure (*runsOn*).

Example 3. The workflow, from our running example (see Section 2.1) can be modelled as instance of the IT-landscape model, consider Figure 3. The activity is associated to five events, $\mathrm{Ev}_{1,\ldots,5}$, which are emitted by nodes in the infrastructure. These nodes are split into static ones, *Authentication Service, XACML-PEP, XACML-PDP, Database Service*, and a variable one: *Workstation.* Some information may be static, i.e. non-mutable DNS names for Windows Domain Controllers, but other information isn't, e.g. changing IPs. Therefore the

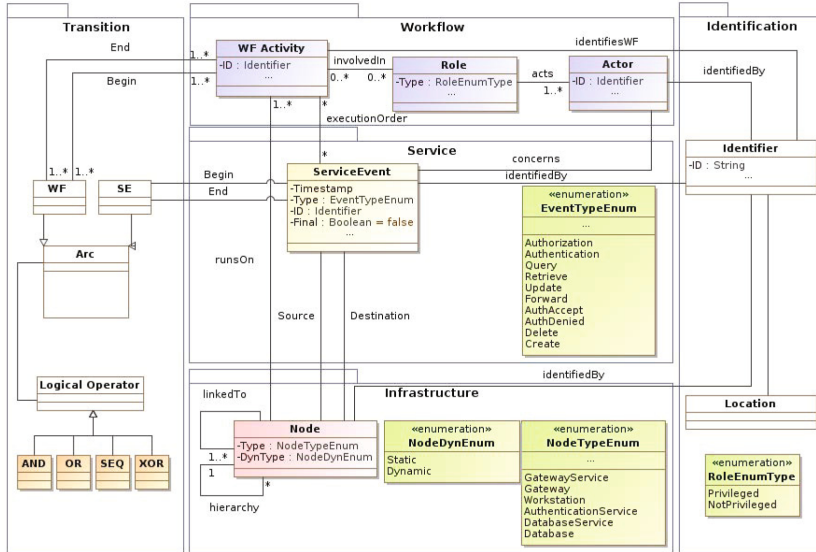


Fig. 2. The IT-Landscape Metamodel

model admits variable ("DynType = Variable") nodes. This allows then to represent variable information, certain only at runtime, in the workflow models. Thus, consiering the example, any node that is flagged as a *Workstation* may partake in the authorization workflow.

3.2 Complex Event Processing

To monitor proper execution of workflows, e.g. to prevent attacks as discussed in Section 2.2, proper rules are needed. Since we are in the comfortable position of knowing how workflows should be executed (sequences), and who is responsible for certain actions we can automatically create CEP rules to monitor these workflows in regard of security. We use the Esper Query Language (EQL)[8] in combination with the Esper CEP engine since it provides several "must have" features, it is open source (GPL GNU Public License v2.0), has an active community and has been shown to outperform competitors, e.g. Drools, ILOG JRules, Amit, for several benchmarks [18].

Translation. During the modelling phase expected events, *ServiceEvent* (see Figure 3), are created and associated to workflow activities. From these models we extract query rules to configure the CEP engine as follows,

1. For each expected event E create an Esper rule.

[8] `http://esper.codehaus.org/`, Accessed: July 20, 2012.

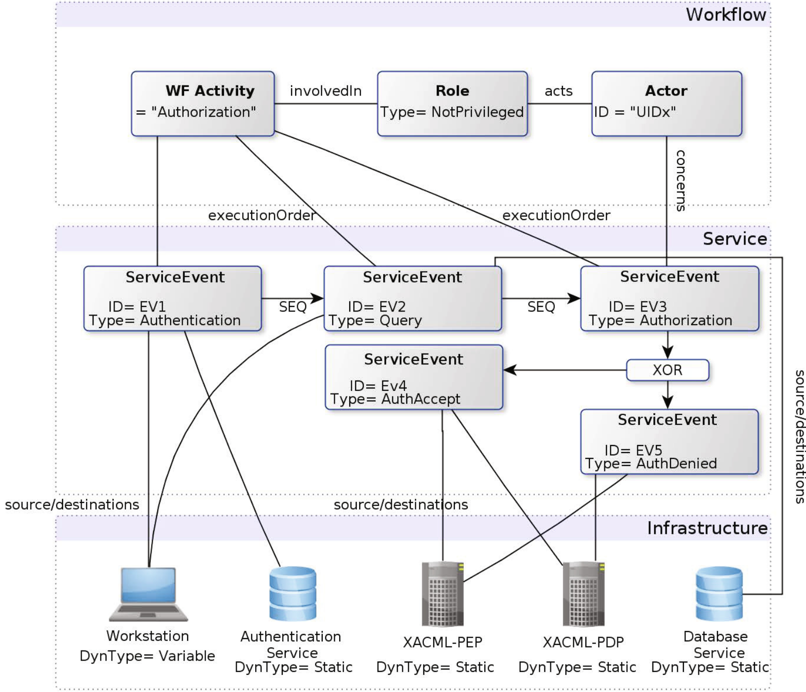


Fig. 3. Sample model for an access control activity in our running example

2. If an event E_n is a successor (SEQ) of another event E_{n-1} then the rules representing those events are conjoined by the EQL keyword "$\rightarrow$". To disambiguate the EQL keyword from the logical implication sign, we denote it as $\underset{seq}{\longrightarrow}$. The formula $Ev_0 \underset{seq}{\longrightarrow} Ev_1$ is only satisfied if and only if Ev_0 is emmitted before Ev_1.
3. If after event E follows either E_0 or E_1 (OR branch) then those events are conjoined by the logical operator OR. Since there is no XOR operator we break it down into a corresponding rule by only using AND, OR and NOT operators.

Applying this method we can carry out basic workflow compliance detection based on service events.

Example 4. The access control specification from Figure 3 would lead to a series of rules for the CEP engine. As a reminder, the model is fulfilled if all events are registered in the same sequence, containing the same meta-data (e.g., the same source), and have not been replicated. Assume we have derived the rules $R_{0,\ldots,5}$ for each expected event. No alert is raised if the formula $R_0 \underset{seq}{\longrightarrow} R_1 \underset{seq}{\longrightarrow} R_2 \underset{seq}{\longrightarrow} R_3 \underset{seq}{\longrightarrow} (R_4 \wedge \neg R_5) \vee (\neg R_4 \wedge R_5)$, which represents the trace of the workflow,

is satisfied.[9] To notice misbehavior in the workflow service events are fed to the CEP engine. The CEP engine then compares entries of the event to the rules and thereby inspects if said events correspond to the expected flow. This way, replay attacks, an altered service invocation sequence, and missing events, are prevented/detected (see Section 2.2).

3.3 Profiling via Clustering

Defining rules for network behavior (i.e. flow behavior) in a fine grained fashion is tough since rules have to consider the complexity of packet-handling, e.g. flags of TCP packets (which are system implementation specific) [19], or the state of congestion in the network [20]. In this section we describe how we link infrastructure events to service calls to enable network and database access related profiling. Figure 4 summarizes how the layers are related. Workflow activities are linked to service call sequences. Then, infrastructure events are linked to service calls. Said events are then clustered to attain the profiles.

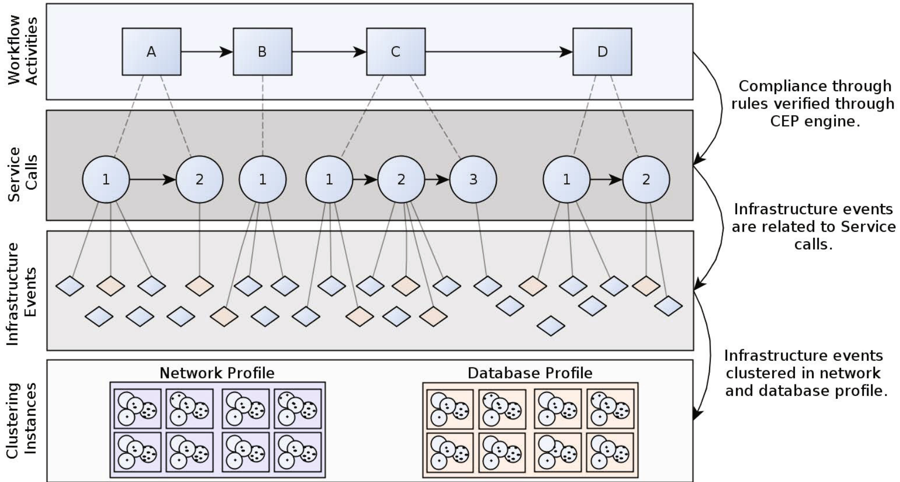


Fig. 4. Overview of connecting the layers

Events Put into Relation. The objective is to sum up network and database behavior for nodes during service calls, and by that determine anomalies. To do this we first have to relate infrastructure events to service events. For that we make use of the following heuristic:

i. Set time interval $t = 10$ seconds. Capture all events during interval t, let the number of events captured during the time interval be $cpt(t) = n$. The applied heuristic finds an interval t' that captures at least half of all events captured within t. Hence we identify $\arg\min_{t' \in [0,10]} cpt(t') > \frac{n}{2}$. This heuristic

[9] For the sake of brevity we substituted AND to "$\wedge$" and NEG to "$\neg$".

gives an interval that subsumes events that have, presumably, a high degree of relatedness.

ii. Relate all network events from a node to the last service call from that same node within t'.

iii. Relate all database CRUD events happening during t', that involve the same identifier that was present in the service call to it.

Related events provide the possibility to create profiles of a service call type, e.g. a *network profile* and a *database profile* (consider Figure 5). As we will discuss in-depth in the following sections we obtain profiles via clustering [21], which makes use of the inherent structure of data. Data instances are grouped (clustered) by common attributes and a similarity measure. The network profile of a service call type, e.g. authentication, contains for example a clustering instance for each node in the network. In contrast, the database profile contains a clustering instance for each user instead of nodes and pinpoints rare user behavior in relation to a service call type. The profile types, depicted in Figure 5, were chosen to address the problem of inferring database and network behavior during workflows in health-care domains.[10]

Network Communication and Database Features. Each network communication and database CRUD event needs to be compared to other instances to determine their relatedness. Therefore we define several simple and aggregated

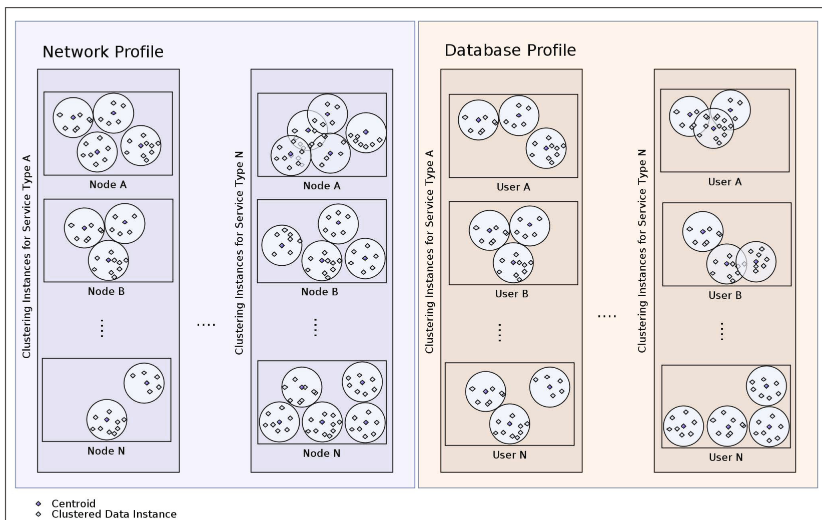


Fig. 5. Visualisation of the network and database profile based on clustering

[10] In other domains other profile types have precedence over the ones we chose. In a nuclear power plant for instance, an anomaly profile might be constituted of temperature sensors, and position of nuclear power rods.

features, some being continuous and others being scaled nominally. The collection of network features and database features form vectors that can be compared through some distance measure. In the network profile the objective is to determine *who is talking to whom* and *how the communication pattern for a service call looks like*. We model TCP/UDP communications based on Hernandez et al. [22].

A TCP/UDP connection can be represented as a flow sequence $(f_0, \ldots, f_{n-1})$ where n is the number of exchanges. Each f_i is a flow triplet $f_i = (a_i, b_i, t_i)$ that describes the i-th communication flow (amount of data). a_i, b_i is the number of bytes from an initiator node to the destination node and vice versa. The time parameter t_i represents the time from f_i to f_{i+1}. It does not make sense to record the time in a more granular fashion, since it depends on the state of the network, e.g. on the degree of congestion [20, 22]. For details on the extraction methods of the flow sequence consider the solution provided in [22]. Moreover our approach uses measurement data gathered with *flowd*.[11]

Each flow sequence of a connection represents a communication pattern. Together with the two participating IPs and ports a sequence serves as base to derive feature vectors that are used to compute the distance to other flows via a distance or correlation measurement. The network feature vector consists of the number of flows in the flow vector, the total number of bytes transmitted (a_{tot}, b_{tot}), the maximum and minimum bytes $(a_{min/max}, b_{min/max})$, the amount of time t_{tot}, the mean bytes (a_μ, b_μ) of the flow sequence, the sample standard deviation (a_σ, b_σ), source IP, destination IP, source port, and destination port.

Database events don't need to be preprocessed as network events since each event is a self-contained action. Features that we extract contain a user id, the object id being accessed, the type of the CRUD operation, the originating host, and the time of the event.

Example 5. A snapshot of the beginning of an FTP communication between node A and B (ports $5320, 21$) produces the following sequence, $((100, 100, 0.2), (1000, 10, 0.2), (22390, 100, 0.2))$. From this sequence the following feature vector $\mathbf{x}$ is created $(3, 23490, 210, 0.6, 7830, 70, 12617, 51.96, \mathrm{IP}_A, \mathrm{IP}_B, 5320, 21)$ that now serves as input for a distance measure.

Clustering and Distance Measurement. Vectors $\mathbf{x}_i = (x_{i0}, \ldots, x_{in-1})$ need to be compared to determine if there are similarities. To compare (measure the distance) various metrics have been proposed [21–23]. For network feature vectors we make use of the standard Euclidean metric, mainly because other authors produced good results with such a measurement [2] in similar domains.

$$d_2(\mathbf{x}_i, \mathbf{y}_i) = \left(\sum_{k=0}^{n-1} (\mathbf{x}_{ik} - \mathbf{y}_{ik})^2 \right)^{\frac{1}{2}} \tag{1}$$

Database feature vectors mostly consist of nominal values which disqualify the use of d_2. In order to cope with this situation we use a distance measure that is used in symbolic data analysis, *objective dissimilarity* [24].

[11] `http://www.mindrot.org/projects/flowd/`, Accessed: July 20, 2012.

$$d_{od}(\mathbf{x}_i, \mathbf{y}_i) = \sum_{k=0}^{n-1} d'_{od}(\mathbf{x}_{ik}, \mathbf{y}_{ik}), \quad d'_{od}(c, c') = \begin{cases} 1 & \text{if } c \neq c' \\ 0 & \text{else} \end{cases} \tag{2}$$

Three issues in distance measuring need to be resolved, the first one being normalization. Assume feature vectors $\{(2, 1222, 5), (5, 1020, 3)\}$, column two disproportionally adds to the distance in d_2. Therefore we normalize each entry in the vectors according to the standard deviation of the vector. The second issue are nominal features in network activity, i.e. source and destination, which cannot be measured using d_2. To cope with that we add weighted constants to the distance between vectors (without the nominal features) where nominal features vary. For instance, two TCP flow sequences $(\mathbf{x}, \mathbf{y})$ may have very similar flows but if the ports and addresses differ, their distance will be affected. The third issue is that objective dissimilarity is not accurate enough to handle continuous values among nominal ones, i.e. the "time" feature in database events. To tackle that, we group daytime seconds to 12 intervals, $[0, 2]_0, [2, 4]_1, \ldots,$[12] which leads for instance to 3.24 a.m. being categorized to interval 1, and 5.44 p.m. to interval 9. Finally the difference of the intervals from the database feature vectors is added to the measured distance of d_{od}, for instance $|t_i - t_j|$.

Clustering Algorithm and Training. The distance measurement we use allows various clustering algorithms, e.g. agglomerative-hierarchical clustering to produce dendrograms, prototype-based clustering, partial clustering, or graph-based clustering [21]. We tried several algorithms and came to the conclusion that a prototype-based clustering (i.e., center-based clustering) best fits to our needs.[13] We employ fixed-width clustering, proposed by [2] that leverages single-linkage clustering. The benefit of using single-linkage clustering in contrast to, e.g. k-means, is a better runtime complexity (it computes clusters with just a single passage through the data instances). Since our framework uses several clustering instances and aggregated data is large it is essential that clustering and detection is fast. In fixed-width clustering [25], clusters have a maximal width, ω, and a cluster center, centroid. Feature vectors outside, either create their own cluster or are assigned to another cluster. Some clusters contain more, some contain less data instances. The assumption that very few instances are real outsiders (η% of data instances) tells us that small clusters, smaller then some threshold T, represent anomalous data points. This is captured by the following formula (3), where $\#(C)$ is the number of data instances within cluster C.

$$\frac{\#(C)}{\sum_{C' \in S} \#(C')} < T \tag{3}$$

We leverage the distance notation from formulas 1 and 2 to $d_2(\mathbf{x}_i, C)$ and $d_{od}(\mathbf{x}_i, C)$ respectively to denote the distance from some vector to a cluster

[12] We abbreviate the intervals from $[n * 3600, (n + 2) * 3600]$ to $[n, n + 2]$, where 3600 seconds are one hour.

[13] Further experiments will show if DBSCAN can be used to create a partial clustering to reduce noise [21].

(represented by its centroid). Formally, the algorithm consists of 3 steps, shown in Figure 6.

1. The set of clusters S is initialized $S = \emptyset$.
2. From the training set a feature vector $\mathbf{x}_i = (\mathbf{x}_{i0}, \ldots, \mathbf{x}_{in-1})$ is retrieved.
   IF S is still empty, $\mathbf{x}_i$ will spawn a new cluster X and is added $S \cup X$. $\mathbf{x}_i$ will be the centroid of X.
   ELSE The cluster C is identified with $\arg\min_{C \in S}(d_2(\mathbf{x}_i, C))$
   IF $d_2(\mathbf{x}_i, C) \leq \omega$, $\mathbf{x}_i$ is added to C.
   ELSE a new cluster is created $S \cup X$.
3. Repeat step 2 until all feature vectors in the training set have been dealt with.

Fig. 6. Basic Clustering Algorithm

Detection of Abnormal Behavior. The detection is straight-forward, a communication flow between nodes via TCP/UDP is translated to a feature representation $\mathbf{x}$ as defined in this section. Then the task is simply to find the cluster C that contains the nearest centroid that does not exceed ω: $\arg\min_{\forall C \in S}(d_2(\mathbf{x}, C)) \leq \omega$. The flow will be classified (and labeled) according to the cluster it was assigned to. Hence, if a cluster was anomalous, then so is the event stream. If $\mathbf{x}$ exceeds the width of the nearest cluster a new anomalous cluster is created.

Example 6. The suspicious database event (Ev_7) that was part of the first infrastructure attack example in Section 2.2 produces an alert, here we show why. In the training phase, database events are captured and associated to the corresponding service call type, here *Authorization Denied*. The *user* feature of the events determines their appropriate clustering instance in the database profile. Since the events that were clustered for UID_x proved to be very similar according to d_{od}, only one cluster C with centroid c = {DC, UID_x,PMR_{12}, Create, 10.00 a.m.} and a predefined $\omega = 4$ was created. While in detection mode a new database event, Ev_7, at 2.02 a.m. is captured. As explained in Section 3.3 a feature representation $\mathbf{x}$ is created, and the distance between C is measured: $d_{od}(\mathbf{x}, C) = 6$. Since the distance exceeds ω, the point $\mathbf{x}$ creates a new cluster and is immediately flagged anomalous because the cluster size falls below the threshold, see Formula 3. A closer look at Ev_7 reveals that it is an update PMR event (crUd) with identifier UID_x and object PMR_{12} at 2.02 a.m. This is an uncommon occurrence. The second infrastructure example from Section 2.2 is solved in a similar fashion, this time, the *network profile* is the source of the alert. There is only one cluster C with centroid $\mathbf{c}$ such that $d_2(\mathbf{x}, c) \leq \omega$. This means that the communication of DC with D is indeed very similar to the communication represented by the cluster C. Since we also add weighted constants to represent differences of nominal values, e.g. IP addresses, the resulting distance exceeds ω. Therefore, the communication pattern creates a new cluster and represents an anomalous event.

Thoughts on False-Positives and Security Testing. Anomalous elements are candidates for misconfiguration, attacks, or non-compliant system usage, i.e. true positives, but may also be false-positives. To reduce the number of false-positives, the threshold, see Formula 3, can be decreased to allow near-empty clusters to be normal. Cluster-widths may also be decreased, which leads to sparse, large, clusters. Another way to mitigate this challenge is to define (a) white lists that allow clusters to be classified as not not harmful, e.g. communication to known non-harmful websites, (b) a threshold $\chi\%$ for the number of infrastructure anomalies that have to be detected in order for an alert to be raised, and (c) test cases that validate clusters. As anomalies on the infrastructure level are traceable to service calls and workflow activities in our approach, test cases for anomalous clusters could be generated. We consider the generation of test cases for the detection of false-positives as future work.

4 Related Work

In this section we discuss related work in the areas of modelling, workflow compliance and anomaly detection via clustering in intrusion detection.

Modelling and CEP. Our metamodel allows the inclusion of roles, node hierarchies, and consists of three abstract layers. This is derived from the work of [14, 15]. Both approaches provide a model-based approach together with concepts and methods for security management. The latter work also includes the *Living Models* approach where models are subjected to constant evolution. Modeling languages have been proposed to create service infrastructures, for instance the *Service Oriented Architecture Modeling Language (SoaML)*[26] and the *Service Markup Language* SML [27]. The two reasons that prohibited the use of SoaML were (i) it is especially designed for modelling SOA services. Our objective lies in services monitoring, yet SoaML provides a too rich vocabulary. Since our interest is mainly event oriented, our metamodel is built to reflect this by leveraging and adapting the idea of event-driven process chains (EPC), deeply discussed in [28], to our needs. The metamodel allows the definition of event-sequences, which allow to detect deviations of workflows. Workflow compliance has been discussed in various publications. Mulo et al. [17] propose monitoring compliance of business processes in SOA via complex event processing (CEP) means. A service invocation is regarded as an event and business process activities as event-trails. These event-trails guide the creation of queries which a CEP engine uses to identify and monitor business activities. Since the business activities are rendered identifiable it is possible to monitor the flow of a business process at runtime, hence, it is possible to detect anomalous process executions. Baresi et al. [29] and Erradi et al. [30] focus on monitoring the execution of centrally orchestrated web services compositions (specified in WS-BPEL). Our work distinguishes itself from the former and the latter in that we also take into account infrastructure anomalies via machine learning, have a strict focus on security and present a modeling environment for said workflows.

Anomaly Detection. Related work in anomaly detection, especially in the area of intrusion detection, is plenty, consider [1] for a survey of different methods. Since unsupervised learning, e.g. clustering, has been used by multiple authors [2, 23, 25, 31] shows the versatility of clustering. Portnoy et al. [2] uses cluster analysis successfully to detect attacks in the KDD 1999 data set.[14] Gu et al. [23] also leverage clustering techniques successfully to detect botnets using their tool "Botminer". Clustering is itself also subject of ongoing research. [25] introduce adaptive clustering to reduce time-based bias in dynamic networks, i.e., traffic variance over time. [31] improves clustering for NIDS by using a density-based clustering algorithm and a grid-based metric. Both [25] and [31] evaluate their efforts on the KDD 1999 data set. We leverage common TCP features and flow modelling [22, 23], common clustering algorithms and metrics [2, 21, 24] (see Section 3). Shared algorithms or flow modelling are only present superficially. Differences include, for instance, the area of anomaly detection. Our method inspects service call profiles for anomalies, instead of, for instance, explicitly detecting botnet activity [23].

To the best of our knowledge, there is no model-driven approach that allows CEP extraction and anomaly detection to monitor (a) the execution of workflows and (b) detect infrastructure aberrations relative to said workflows. As a by-product of the IT-landscape model, and also in contrast to existing work, outlying events can be used to locate suspicious nodes, services and workflows.

5 Conclusion and Future Work

We have presented an anomaly detection framework that allows real-time monitoring of critical workflows. In contrast to existing monitoring work we consider the interplay of multiple layers at once and can, thus, link infrastructure anomalies to workflows. Workflow aberrations are detected by CEP rules, which are extracted from models, anomalies are inferred by network and database profiles (which are related to service calls). Summarizing this approach enables to detect workflow aberrations via CEP and connected anomalies, e.g. anomalous CRUD activities. Future work consists in the continuation of the implementation and evaluation efforts to create a fully functional anomaly detection framework that follows the scheme described in this paper. The framework as a whole will be evaluated based on a real-world setting.

Besides the evaluation of the overall approach, future work will consist of fine-tuning metrics, features, and clustering algorithms in order to better cope with, e.g. nominal values, overlapping clusters. A promising approach to measure the distance between nominal values are *data generalization hierarchies* [32]. As already mentioned before, we will also consider the generation of test cases for the detection of false-positives.

[14] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, Accessed: July 20, 2012.

References

1. Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E.: Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. Computers & Security 28(1-2), 18–28 (2009)
2. Portnoy, L., Eskin, E., Stolfo, S.: Intrusion detection with unlabeled data using clustering. In: Proceedings of ACM CSS Workshop on Data Mining Applied to Security (2001)
3. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41(3), 15:1–15:58 (2009)
4. Eckert, M., Bry, F.: Complex Event Processing, CEP (2009)
5. OMG: Omg uml specification, v2.0 (2005)
6. Moses, T.: eXtensible Access Control Markup Language TC v2.0 (XACML) (2005)
7. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and System Security (TISSEC) 2(1), 65–104 (1999)
8. Walker-Morgan, D.: Vsftpd backdoor discovered in source code. Website (2011), `http://h-online.com/-1272310` (accessed: July 20, 2012)
9. Hoglund, G., Butler, J.: Rootkits: subverting the Windows kernel. Addison-Wesley Professional (2006)
10. Peikari, C., Chuvakin, A.: Security Warrior. O'Reilly (2004)
11. Wells, J.: Computer fraud casebook: the bytes that bite. John Wiley & Sons Inc. (2008)
12. Ye, N., Emran, S.M., Chen, Q., Vilbert, S.: Multivariate Statistical Analysis of Audit Trails for Host-based Intrusion Detection. IEEE Transactions on Computers 51(7), 810–820 (2002)
13. Roesch, M.: Snort: Lightweight intrusion detection for networks. In: LISA, pp. 229–238. USENIX (1999)
14. Breu, R., Innerhofer-Oberperfler, F., Yautsiukhin, A.: Quantitative assessment of enterprise security system. In: The Third International Conference on Availability, Reliability and Security, pp. 921–928. IEEE (2008)
15. Innerhofer-Oberperfler, F., Breu, R., Hafner, M.: Living security collaborative security management in a changing world. In: Parallel and Distributed Computing and Networks/720: Software Engineering. ACTA Press (2011)
16. Xtext, `http://www.eclipse.org/Xtext/` (accessed: July 20, 2012)
17. Mulo, E., Zdun, U., Dustdar, S.: Monitoring web service event trails for business compliance. In: 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–8. IEEE (2009)
18. Grohe, S., Schlameu, C., Sommer, R.: Performancevergleich von cep-engines. Technical report, Hochschulschriftenserver der Universitt Stuttgart (Germany) (2010), `http://elib.uni-stuttgart.de/opus/oai2/oai2.php`
19. McClure, S., Scambray, J., Kurtz, G.: Hacking exposed 6. McGraw-Hill (2009)
20. Allman, M., Paxson, V., Stevens, W.: RFC 2581 (rfc2581) - TCP Congestion Control. Technical Report 2581 (1999)
21. Tan, P., Steinbach, M., Kumar, V.: Cluster Analysis: basic concepts and algorithms. In: Introduction to Data Mining, Addison-Wensley (2006)
22. Hernandez-Campos, F., Nobel, A.B., Smith, F.D., Jeffay, K.: Understanding patterns of tcp connection usage with statistical clustering. In: 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 35–44. IEEE (2005)

23. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection. In: Proceedings of the 17th Conference on Security Symposium, pp. 139–154 (2008)
24. Malerba, D., Esposito, F., Gioviale, V., Tamma, V.: Comparing dissimilarity measures for symbolic data analysis. In: Proceedings of Exchange of Technology and Know-how and New Techniques and Technologies for Statistics, vol. 1, pp. 473–481 (2001)
25. Oldmeadow, J., Ravinutala, S., Leckie, C.: Adaptive Clustering for Network Intrusion Detection. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 255–259. Springer, Heidelberg (2004)
26. Berre, A.: Service oriented architecture modeling language (soaml)-specification for the uml profile and metamodel for services (upms) (2008)
27. Popescu, V., Smith, V., Pandit, B.: Service modeling language, version 1.1. W3C recommendation, W3C (May 2009), http://www.w3.org/TR/2009/REC-sml-20090512/
28. van der Aalst, W.: Formalization and verification of event-driven process chains. Information and Software Technology 41(10), 639–650 (1999)
29. Baresi, L., Guinea, S., Plebani, P.: WS-Policy for service monitoring. In: Technologies for E-Services, pp. 72–83 (2006)
30. Erradi, A., Maheshwari, P., Tosic, V.: WS-Policy based monitoring of composite web services (2007)
31. Leung, K., Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the Twenty-eighth Australasian Conference on Computer Science, vol. 38, pp. 333–342 (2005)
32. Julisch, K.: Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security (TISSEC) 6(4), 471 (2003)

A Hierarchical Variability Model for Software Product Lines*

Dilian Gurov[1], Bjarte M. Østvold[2], and Ina Schaefer[3]

[1] Royal Institute of Technology, Stockholm, Sweden
dilian@csc.kth.se
[2] Norwegian Computing Center, Oslo, Norway
bjarte@nr.no
[3] TU Braunschweig, Germany
i.schaefer@tu-bs.de

Abstract. A key challenge in software product line engineering is to represent solution space variability in an economic, yet easily understandable fashion. We introduce the notion of hierarchical variability models to describe families of products in a manner that facilitates their modular design and analysis. In this model, a family is represented by a *common* set of artifacts and a set of *variation points* with associated variants. A variant is again a hierarchical variability model, leading to a hierarchical structure. These models, however, are not unique with respect to the families they define. We therefore propose a quantitative measure on hierarchical variability models that expresses the degree to which a variability model captures commonality and variability in a family. Further, by imposing well-formedness constraints, we identify a class of variability models that, by construction, have maximal measure and are unique for the families they define. For this class of *simple families*, we provide a procedure that reconstructs their hierarchical variability model. The reconstructed model can be used to drive various static analyses by divide-and-conquer reasoning. Hierarchical variability models strike a balance between the formalism's expressiveness and the desirable property of model uniqueness. We illustrate the approach by a small product line of Java classes.

1 Introduction

System diversity is prevalent in modern software systems. Systems simultaneously exist in many different variants in order to comply with different requirements. Software product line engineering [18] aims at developing a family of system variants by managed reuse in order to decrease time to market and to improve quality. The variability of the different products in a software product line can be represented at different levels [7]. *Problem space variability* describes

* Partly funded by the EU project *HATS*, Highly Adaptable and Trustworthy Software using Formal Models (FP7-231620) and the Deutsche Forschungsgemeinschaft (SCHA1635/2-1).

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 181–199, 2012.
© Springer-Verlag Berlin Heidelberg 2012

product variation in terms of features where a feature is a user-visible product characteristic. The set of valid feature configurations defines the set of possible products. However, features do not relate to the actual artifacts that are used to realize the products.

Problem-space variability, based on features, is at the requirements level, while *solution space variability* is at the design and implementation level. Solution-space variability describes product variation in terms of shared artifacts that are used to build the actual products of the product line. In this paper, we capture solution space variability in terms of variable artifact implementations for fixed artifact names. This means that in different product variants an artifact with the same name can be realized with different implementations. Here, an artifact can be a component at a suitable level of granularity, such as a method, a class, or a module. Then, the artifact name would be the method signature (including the method name), interface signature or module signature, respectively, while the artifact implementation would be the method body, interface implementation, or the module realization. Previously, we used the finest of these levels [20], i.e., method signatures and method bodies, while in Section 4 we show another interpretation, where artifact names are types and artifact implementations are classes, interfaces or types implementing the former type.

In order to describe the relationship of the artifact names to the artifact implementations in the product variants, we introduce *hierarchical variability models*. Hierarchical variability models represent, in a hierarchical manner, the artifact implementations that are common to all products and the variations in the artifact implementations that can occur between different products. On each hierarchical level, there is a *common set* of artifact implementations that represent parts shared by all products, while *variation points* represent parts that can vary from product to product. Every variation point is associated with a set of variants that represent choices for realizing the variation point in different ways. A variant is itself represented by a hierarchical variability model, introducing a new level of hierarchy. A product described by a hierarchical variability model is obtained by selecting a variant at every variation point.

We have previously argued that hierarchical variability models support modular design [12] and divide-and-conquer reasoning for product lines, such as the formal verification of critical requirements of all products of a family [20]. In general, given a concrete program analysis, factoring out common implementations naturally reduces redundancy in the analysis. At variants with more than one variation point, the analysis problem is *decomposed* into simpler subproblems (since they expose orthogonality), while at variation points with more than one variant, the same problem is solved *independently* as a case analysis (since they don't share implementations). Thus, a hierarchical variability model can be seen as a (divide-and-conquer style) scheme for decomposing and splitting an analysis over a family.

In this paper, we propose a hierarchical variability model that is *simple* in the sense that it requires the choice of exactly one variant for every variation point, and does not specify any constraints between choices made at different variation

points. Figure 1 shows a simple hierarchical variability model for a cash desk system, depicted as a tree with a root node marked CashDesk. Common to all cash desk systems are the following artifact implementations: $saleProcess_{\mathsf{CashDesk}}$ for handling the sale process, and two implementations called $writeReceipt_{\mathsf{CashDesk}}$ and $updateStock_{\mathsf{CashDesk}}$ responsible for the corresponding tasks. The notational convention is that an artifact implementation is an artifact name (*e.g.*, *saleProcess*) with an index (*e.g.*, CashDesk).

At the first level of hierarchy, a cash desk can vary in two uncorrelated (*i.e.*, orthogonal) aspects. First, there are two methods to input data about merchandise payed for at the cash desk: by keyboard or using a scanner. Second, there are two ways to pay, either in cash or by card. Thus, on the first level, the hierarchical variability model in the figure has two variation points: InputMethods and PaymentMethods. Each variation point has associated variants which capture one particular way of realizing the variation point. Variation point InputMethods has two variants, Keyboard and Scanner, each with an implementation of the corresponding input method. Variation point PaymentMethods also has two variants: Cash and Card for the two forms of payment. Both variants provide an artifact named *slot* for inserting the means of payment and *pay* for the actual payment process with different implementations. *slot* has one implementation in each variant, whereas *pay* has one implementation for cash and three variant implementations for card, corresponding to three different types of card.

The intention of a hierarchical variability model is that, on each level of hierarchy, *common sets* of artifact implementations are factored out, while *uncorre-*

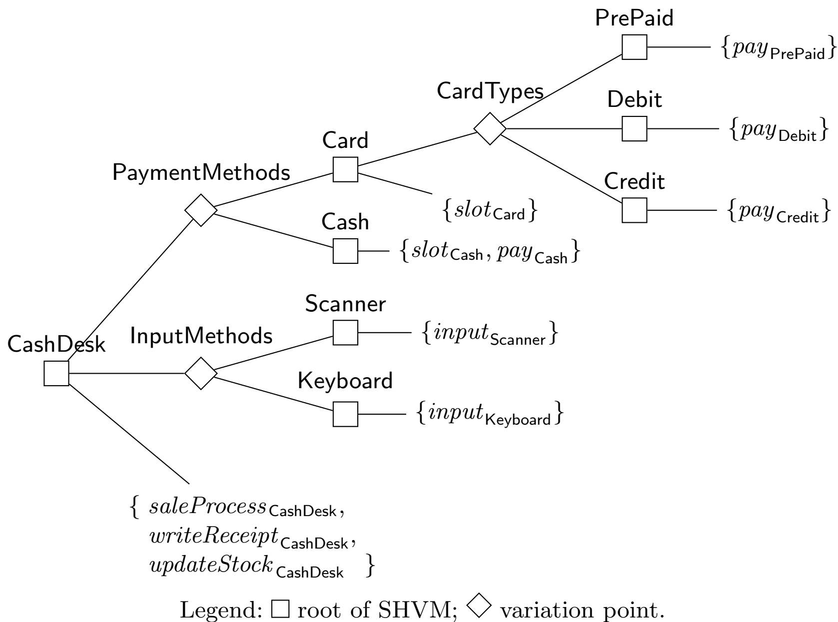


Fig. 1. The CashDesk hierarchical variability model (drawn sideways)

lated (or *orthogonal*) sets of artifact implementations are delegated to different variation points. To provide a measure for the quality of hierarchical variability models for defining a family in an economical way, we define the *separation degree* of a model (Definition 6) as the ratio between the total number of artifact implementations from which products are constructed and the total number of artifact implementation occurrences in the common sets of the model. Thus, high-quality models capture repetitions across products in a family without repetition in the model. The maximal possible separation degree of one is reached in models where every artifact implementation occurs in exactly one common set.

In order to reason formally about hierarchical variability models, we provide these with a formal semantics in terms of the products that can be generated by variant selection. We define *well-formedness constraints* on hierarchical variability models, under which the separation degree of the model is equal to one by construction. We term the class of families generated by well-formed variability models *simple families*, and define this class in a model-independent fashion. We present a transformation from simple families to hierarchical variability models that (re)constructs the unique well-formed model that generates the family. Uniqueness is established by showing that the two transformations—from well-formed models to simple families and *vice versa*—are inverses of each other. For practical purposes, the latter transformation can be used for *variability model mining* from a given family of products. Simple hierarchical variability models thus strike a balance between the expressiveness of the modeling formalism—no bindings and being grammar-like—and the desirable property of uniqueness of models: With a more expressive modeling formalism uniqueness may not be achievable.

To the best of our knowledge, this work is the first to provide a formal semantics for hierarchical variability models in the solution space, and to characterize a class of variability models through the class of generated product families. Previous work has been informal, as for instance the Koala component model [22], hierarchical variability modeling for software architectures [12], or plastic partial components [17]. Our work is also the first to provide a technique for constructing a hierarchical solution space variability model from a given family.

Our main *contributions* are thus:

(i) A formal definition of *simple families* as families that can be formed from artifact implementations by using a set of base operations on families (Section 2.1).
(ii) A definition of *simple hierarchical variability models*, together with a quality measure called *separation degree* and a set of *well-formedness constraints* yielding (by construction) models with maximal measure (Section 2.2).
(iii) A formal semantics for hierarchical variability models in terms of *family generation*, and a proof that, for every well-formed variability model, the generated family is simple (Section 3.1).
(iv) A procedure to construct hierarchical variability models from simple families that produces well-formed models (Section 3.2).
(v) A *characterization result* stating that, for well-formed hierarchical variability models and simple families, family generation and hierarchical variability

model construction are *inverses* of each other, thus implying correctness of model construction and uniqueness of well-formed models with respect to the families they generate (Section 3.3).

All proofs and some supporting results can be found in the accompanying technical report [11].

2 Families and Variability Models

In this section, we present product families as a semantic domain for our hierarchical variability model. The model is presented in the following subsection.

2.1 Families

We consider products realized by a set of artifact implementations for a given set of artifact names. An artifact can be thought of as, *e.g.*, a component or a method. We fix a countably infinite set of artifact names Art.

Definition 1 (Product, family). *An* artifact implementation *is an indexed artifact name; let a_i denote the i-th implementation of artifact name a. A* product *P is a finite set of artifact implementations, where for each artifact name there is at most one implementation. A* family *$\mathcal{F}$ is a finite non-empty set of products.*

Thus, products can be seen as partial maps from artifact names to natural numbers, having a finite domain; we use Nat^{Art} to denote the set of all products over Art. We refer to singleton set families as *core* families, or simply cores. The family consisting of the empty product is denoted $1_{\mathcal{F}}$.

Example 1. Here are some families that are used later to illustrate various notions.

$$\begin{aligned}\mathcal{F}_A = \{ \; & \{a_1, b_1, c_1, d_1, e_1\}, \{a_1, b_1, c_1, d_1, e_2\}, \{a_1, b_1, c_2, d_2, e_1\}, \\ & \{a_1, b_1, c_2, d_2, e_2\}, \{a_1, b_1, c_2, d_3, e_1\}, \{a_1, b_1, c_2, d_3, e_2\} \; \} \\ \mathcal{F}_B = \{ \; & \{a_1, b_1\}, \{a_1, b_2\}, \{a_2, b_1\} \; \}\end{aligned}$$

Next, we define two mappings for identifying the artifact names and artifact implementations that occur in a family.

Definition 2 (Family names and implementations). *The mapping* $\text{names}(\mathcal{F})$ *from families to sets of artifact names and the mapping* $\text{impls}(\mathcal{F})$ *from families to sets of artifact implementations are defined as follows, where $a^1, \ldots, a^n \in Art$ and $i_1, \ldots, i_n \in Nat$:*

$$\begin{aligned}\text{names}(\mathcal{F}) &\stackrel{\text{def}}{=} \textstyle\bigcup_{P \in \mathcal{F}} \text{names}(P) \\ \textit{where } \text{names}\left(\{a^1_{i_1}, \ldots, a^n_{i_n}\}\right) &\stackrel{\text{def}}{=} \{a^1, \ldots, a^n\} \\ \text{impls}(\mathcal{F}) &\stackrel{\text{def}}{=} \textstyle\bigcup_{P \in \mathcal{F}} \text{impls}(P) \\ \textit{where } \text{impls}(\{a^1_{i_1}, \ldots, a^n_{i_n}\}) &\stackrel{\text{def}}{=} \{a^1_{i_1}, \ldots, a^n_{i_n}\}\end{aligned}$$

In this definition we abuse notation by also defining mappings with the same names from products to the same co-domains.

We use two binary operations on families, the usual set union operation $\cup$ and the *product union* operation $\bowtie$ over families with disjoint sets of artifact names defined by:

$$\mathcal{F}_1 \bowtie \mathcal{F}_2 \stackrel{\mathsf{def}}{=} \{P_1 \cup P_2 \,|\, P_1 \in \mathcal{F}_1 \wedge P_2 \in \mathcal{F}_2\}$$

and generalized through $\prod_{i \in I} \mathcal{F}_i$ to non-empty sets of families. Intuitively, the product union of two families is the family having as products all possible combinations of products of the original families. Both operations are commutative and associative.

We now define a distinct class of families that we later relate to a specific class of hierarchical variability models. The class of families contains all single-product families consisting of a single artifact implementation, and is closed under product union of families over disjoint sets of artifact names, and under union of families over the same set of artifact names, but having disjoint implementations.

Definition 3 (Simple family). *The class* $\boldsymbol{F}$ *of* simple *families is the least set of families closed under the formation rules:*

($\mathcal{F}$1) $\{\{a_i\}\} \in \boldsymbol{F}$ *for any* $a \in Art$ *and* $i \in Nat$*.*
($\mathcal{F}$2) $\mathcal{F}_1 \bowtie \mathcal{F}_2 \in \boldsymbol{F}$ *for any* $\mathcal{F}_1, \mathcal{F}_2 \in \boldsymbol{F}$ *such that* $\text{names}\,(\mathcal{F}_1) \cap \text{names}\,(\mathcal{F}_2) = \emptyset$.
($\mathcal{F}$3) $\mathcal{F}_1 \cup \mathcal{F}_2 \in \boldsymbol{F}$ *for any* $\mathcal{F}_1, \mathcal{F}_2 \in \boldsymbol{F}$ *such that* $\text{names}\,(\mathcal{F}_1) = \text{names}\,(\mathcal{F}_2)$ *and* $\text{impls}(\mathcal{F}_1) \cap \text{impls}(\mathcal{F}_2) = \emptyset$.

Example 2. The family $\{\,\{a_1, b_1\}\,, \{a_1, b_2\}\,\}$ is simple, as it can be presented as $\{\{a_1\}\} \bowtie (\{\{b_1\}\} \cup \{\{b_2\}\})$ which follows the above formation rules. Family $\mathcal{F}_A$ of Example 1 is also simple (as we shall see later in Example 6), while family $\mathcal{F}_B$ of Example 1 is not: there is no way of building this family with the above formation rules.

Simplicity of families expresses that different functionalities in a product line are always orthogonal, and that alternative realizations of the same functionality have always disjoint implementations. These assumptions are rather heavy and may not always hold in practice. But only under such severe constraints can one hope for such a (strong) uniqueness result as the one obtained later (Section 2.1).

Two distinct artifact names $a, b \in \text{names}\,(\mathcal{F})$ are termed *correlated* in a family $\mathcal{F}$, denoted $a C_{\mathcal{F}} b$, if there are implementations $a_i, b_j \in \text{impls}(\mathcal{F})$ such that no product in $\mathcal{F}$ contains both implementations simultaneously. Otherwise, names a and b are termed *uncorrelated* or *orthogonal*. The correlation relation $C_{\mathcal{F}}$ on $\text{names}\,(\mathcal{F})$ is symmetric, and hence, its reflexive and transitive closure $C^*_{\mathcal{F}}$ is an equivalence relation. As usual, we denote the partitioning induced by $C^*_{\mathcal{F}}$ on $\text{names}\,(\mathcal{F})$ by $\text{names}\,(\mathcal{F})\,/C^*_{\mathcal{F}}$ (quotient set).

Example 3. Consider family $\mathcal{F}_A$ of Example 1. The only two correlated names are c and d, evidenced by the lack of a product containing, for instance, c_1 and d_2. Thus, we have $\text{names}\,(\mathcal{F}_A)\,/C^*_{\mathcal{F}_A} = \{\{a\}, \{b\}, \{c, d\}, \{e\}\}$.

Correlation (and orthogonality) extends naturally to products in a family: Products P and P' are correlated in $\mathcal{F}$ if some artifact name occurring in P is correlated to some artifact name occurring in P'. Similarly, we define the sharing relation $N_{\mathcal{F}}$ on $\mathcal{F}$ as $P_1 \, N_{\mathcal{F}} \, P_2 \overset{\text{def}}{\Leftrightarrow} P_1 \cap P_2 \neq \emptyset$, and use its reflexive and transitive closure $N^*_{\mathcal{F}}$ to partition the family $\mathcal{F}$.

When restricted to simple families, the two operations on families do not distribute over each other. This entails that simple families have *unique* formation trees modulo commutativity and associativity of the two operations associated with the rules.

2.2 Variability Models

In order to represent solution space variability of families in terms of shared artifact implementations, we consider simple hierarchical variability models.

Definition 4 (Simple hierarchical variability model). *A* simple hierarchical variability model (SHVM) $\mathcal{S}$ *is inductively defined as:*

(i) a (possibly empty) common set *of artifact implementations* M_C, *or*
(ii) a pair $(M_C, \{VP_1, \ldots, VP_n\})$ *where* M_C *is defined as above and the set* $\{VP_1, \ldots, VP_n\}$ *of* variation points *is non-empty. A variation point* $VP_i = \{\mathcal{S}_{i,j} \mid 1 \leq j \leq k_i\}$, *where* $k_i \geq 2$, *is a set of (at least two) SHVMs called* variants.

We sometimes refer to an SHVM simply as a variability model. An SHVM with only a common set of artifact implementations is called *ground model*. An SHVM generates a family $\mathcal{F}$ through all possible ways of resolving the variabilities of the SHVM. This process recursively selects exactly one variant for each variation point. We defer a formal definition of such a semantics for SHVMs to Section 3.1. Variability models can be naturally depicted as trees, where leaves are common sets of artifact implementations, and internal nodes are the roots of SHVMs or variation points.

Example 4. Figure 2 and Figure 3 show four variability models named $\mathcal{S}_{A1}$, $\mathcal{S}_{A2}$, $\mathcal{S}_{B1}$, and $\mathcal{S}_{B2}$. In these figures, (sub)trees showing variability models are rooted with boxes, and subtrees showing variation points are rooted with diamonds.

In analogy with Definition 2, we define two mappings for identifying the artifact names and artifact implementations that occur in SHVMs.

Definition 5 (SHVM names and implementations). *The mapping* names$(\mathcal{S})$ *from SHVMs to sets of artifact names and the mapping* impls$(\mathcal{S})$ *from SHVMs to sets of artifact implementations are defined as follows, where* $a^1, \ldots, a^n \in Art$ *and* $i_1, \ldots, i_n \in Nat$*:*

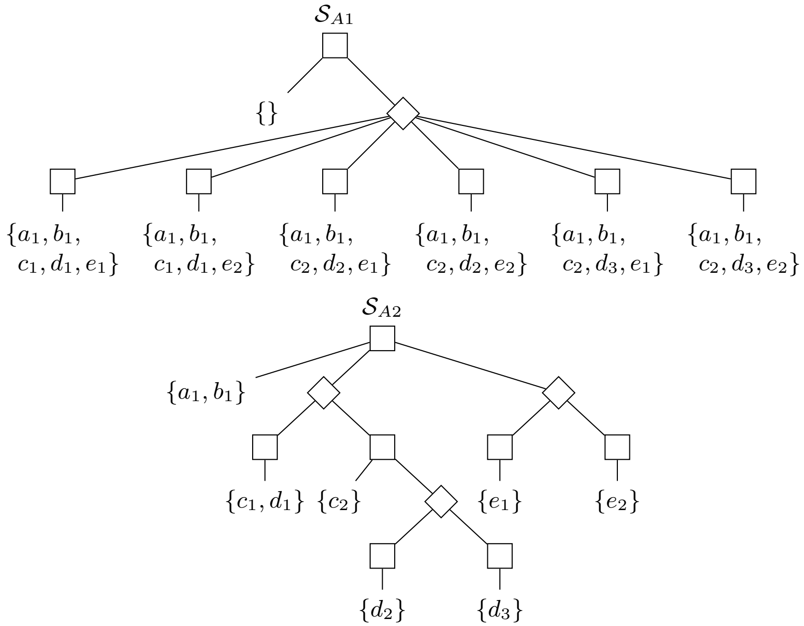


Fig. 2. SHVMs $\mathcal{S}_{A1}$ and $\mathcal{S}_{A2}$ for the family $\mathcal{F}_A$ in Example 1

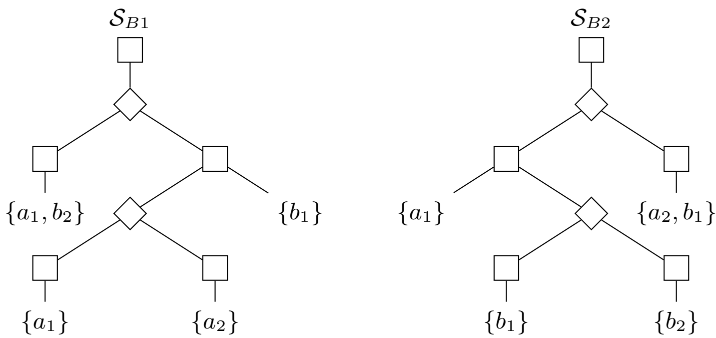


Fig. 3. SHVMs $\mathcal{S}_{B1}$ and $\mathcal{S}_{B2}$ for the family $\mathcal{F}_B$ in Example 1

$$\text{names}\left(\{a^1_{i_1},\ldots,a^n_{i_n}\}\right) \stackrel{\text{def}}{=} \{a^1,\ldots,a^n\}$$
$$\text{names}\left((M_c,\{VP_1,\ldots,VP_n\})\right) \stackrel{\text{def}}{=} \text{names}\,(M_C) \cup \bigcup_{1\le i\le n} \text{names}\,(VP_i)$$
$$\textit{where}\ \text{names}\,(VP) \stackrel{\text{def}}{=} \bigcup_{\mathcal{S}\in VP} \text{names}\,(\mathcal{S})$$
$$\text{impls}(\{a^1_{i_1},\ldots,a^n_{i_n}\}) \stackrel{\text{def}}{=} \{a^1_{i_1},\ldots,a^n_{i_n}\}$$
$$\text{impls}((M_c,\{VP_1,\ldots,VP_n\})) \stackrel{\text{def}}{=} \text{impls}(M_C) \cup \bigcup_{1\le i\le n} \text{impls}(VP_i)$$
$$\textit{where}\ \text{impls}(VP) \stackrel{\text{def}}{=} \bigcup_{\mathcal{S}\in VP} \text{impls}(\mathcal{S})$$

Again we abuse notation by also defining mappings with the same names from variation points to the same co-domains.

Next, we define a measure of the degree of separation in a variability model, as the proportion between the number of artifact implementations of a variability model and the total size of the leaves of the SHVM tree. The separation degree is, thus, a number in the interval $\langle 0,1]$, and captures the degree to which the commonalities and orthogonalities of products are factored out as common sets and variation points in a variability model, respectively: the higher this degree, the less artifact implementations occur repeatedly in more than one leaf. The maximal value of 1 holds when every artifact implementation occurs in exactly one leaf; this is trivially the case for ground models.

Definition 6 (Separation degree). *The* separation degree $sd(\mathcal{S})$ *of a variability model* $\mathcal{S}$ *is defined as:*

$$sd(\{\}) \stackrel{\text{def}}{=} 1$$
$$sd(\mathcal{S}) \stackrel{\text{def}}{=} \frac{|\text{impls}(\mathcal{S})|}{sd'(\mathcal{S})} \qquad \textit{if}\ \mathcal{S} \neq \{\}$$

where $sd'(\mathcal{S})$ *is inductively defined as follows:*

$$sd'(M_C) \stackrel{\text{def}}{=} |M_C|$$
$$sd'((M_C,\{VP_1,\ldots,VP_n\})) \stackrel{\text{def}}{=} sd'(M_C) + \Sigma_{1\le i\le n} sd'(VP_i)$$
$$\textit{where}\ sd'(VP) \stackrel{\text{def}}{=} \Sigma_{\mathcal{S}\in VP} sd'(\mathcal{S})$$

As usual $|S|$ denotes the cardinality of set S.

Intuitively this definition captures the extent to which orthogonal artifact implementations are delegated to separate variation points, and the extent to which disjointness of artifact implementations is delegated to separate variants. Since this is the original intention of variation points and variants in our model, separation degree is an obvious quality measure indicating how well the model is used for the purpose of hierarchically representing a software family (that is, a set of products).

The following definition provides a set of well-formedness constraints on SHVMs.

Definition 7 (Well-formed variability model). *A ground variability model $\mathcal{S} = M_C$ is* well-formed *if constraint ($\mathcal{S}1$) below is satisfied. A variability model $\mathcal{S} = (M_C, \{VP_1, \ldots, VP_n\})$ with variation points $VP_i = \{\mathcal{S}_{i,j} \mid 1 \leq j \leq k_i\}$ is* well-formed *if all variants $\mathcal{S}_{i,j}$ are well-formed, and furthermore, the following constraints are satisfied:*

($\mathcal{S}1$) M_C *implements artifact names at most once.*
($\mathcal{S}2$) $\text{names}(M_C) \cap \text{names}(VP_i) = \emptyset$ *for all* i*, and* $\text{names}(VP_{i_1}) \cap \text{names}(VP_{i_2}) = \emptyset$ *whenever* $i_1 \neq i_2$.
($\mathcal{S}3$) $\text{names}(\mathcal{S}_{i,j_1}) = \text{names}(\mathcal{S}_{i,j_2})$ *for all* i, j_1, j_2*, and* $\text{impls}(\mathcal{S}_{i,j_1}) \cap \text{impls}(\mathcal{S}_{i,j_2}) = \emptyset$ *whenever* $j_1 \neq j_2$.

Example 5. Consider the SHVMs $\mathcal{S}_{A1}$ and $\mathcal{S}_{A2}$ depicted in Figure 2. $\mathcal{S}_{A1}$ is not well-formed whereas $\mathcal{S}_{A2}$ is. The separation degrees are $sd(\mathcal{S}_{A1}) = \frac{9}{6 \cdot 5} = 0.3$ and $sd(\mathcal{S}_{A2}) = \frac{9}{9} = 1$. Figure 3 depicts another two SHVMs, $\mathcal{S}_{B1}$ and $\mathcal{S}_{B2}$. Neither of these are well-formed and both have separation degree $\frac{4}{5} = 0.8$.

The constraints in Definition 6 ensure that the separation degree of a well-formed SHVM is equal to one, and is thus maximal.

Proposition 1. *If variability model $\mathcal{S}$ is well-formed then $sd(\mathcal{S}) = 1$.*

Note that the converse of Proposition 1 does not hold in general: The variability model $M_C = \{a_1, a_2\}$ has separation degree 1, but well-formedness constraint ($\mathcal{S}1$) is not satisfied.

3 Relating Families and Variability Models

In this section, we present translations between well-formed variability models and simple families, and show that they are inverses of each other. In particular, this entails that the translation from simple families to variability models produces the unique well-formed model generating the respective family, thus giving a procedure for constructing a variability model from a given family.

3.1 From Variability Models to Families

The set of products generated by a ground model is the singleton set comprising the set of common artifact implementations (and, thus, representing one product). The set of products generated by a variation point is the union of the product sets generated by its variants. Finally, the set of products generated by an SHVM with a non-empty set of variation points is the set of all products consisting of the common artifact implementations and of exactly one product from the set generated by each variation point.

Definition 8 (Family generation). *The mapping family($\mathcal{S}$) from variability models to families is inductively defined as follows:*

$$family(M_C) \stackrel{\mathsf{def}}{=} \{M_C\}$$
$$family((M_C, \{VP_1, \ldots, VP_n\})) \stackrel{\mathsf{def}}{=} \{M_C\} \bowtie \prod_{1 \leq i \leq n} family(VP_i)$$
$$where\ family(VP) \stackrel{\mathsf{def}}{=} \bigcup_{\mathcal{S} \in VP} family(\mathcal{S})$$

We say that variability model $\mathcal{S}$ generates *$family(\mathcal{S})$.*

Here we again abuse notation by also defining a mapping with the same name from variation points to the same co-domain. Family generation is well-defined in the sense that well-formed variability models generate simple families.

Proposition 2. *If variability model $\mathcal{S}$ is well-formed, then $family(\mathcal{S})$ is simple.*

Example 6. SHVMs $\mathcal{S}_{A1}$ and $\mathcal{S}_{A2}$ in Figure 2 both generate family $\mathcal{F}_A$ in Example 1, implying that family $\mathcal{F}_A$ is simple since $\mathcal{S}_{A2}$ is well-formed. SHVMs $\mathcal{S}_{B1}$ and $\mathcal{S}_{B2}$ in Figure 2 both generate family $\mathcal{F}_B$ in Example 1. Of these four, $\mathcal{S}_{A2}$, $\mathcal{S}_{B1}$ and $\mathcal{S}_{B2}$ have maximal separation degree in the sense that, for each of the families $\mathcal{F}_A$ and $\mathcal{F}_B$, no other SHVMs for the same family have higher separation degree.

3.2 From Families to Variability Models

We now present a reverse transformation from simple families to well-formed variability models. Recall that simple families have unique formation trees modulo commutativity and associativity of the two operations. Well-formed SHVMs can thus be seen as a uniform way of grouping the formation terms. Every family $\mathcal{F}$ can be decomposed into the form:

$$\mathcal{F} = \{P\} \bowtie \mathcal{F}_V, \quad \mathcal{F}_V = \prod_{1 \leq i \leq n} \mathcal{F}_i, \quad \mathcal{F}_i = \bigcup_{1 \leq j \leq k_i} \mathcal{F}_{i,j}$$

where P is a product, or equivalently, as a single equation:

$$\mathcal{F} = \{P\} \bowtie \prod_{1 \leq i \leq n} \bigcup_{1 \leq j \leq k_i} \mathcal{F}_{i,j} \qquad (*)$$

The existence of the decomposition is ensured since every family $\mathcal{F}$ can be trivially decomposed as $\{\emptyset\} \bowtie \prod \bigcup \mathcal{F}$, *i.e.*, with product P being empty and $n = k_1 = 1$. Decomposition $(*)$ is only unique under additional constraints, under which the decomposition is called canonical.

Definition 9 (Canonical form). *A family $\mathcal{F}$, decomposed as equation $(*)$ above, is in* canonical form *if the following conditions hold:*

- *($\mathcal{C}$1) The product P is the set of artifact implementations that are common to all products in $\mathcal{F}$.*
- *($\mathcal{C}$2) The set of artifact names in $\mathcal{F}_V$ has n equivalence classes w.r.t. correlated artifact names $C^*_{\mathcal{F}_V}$, and for the i-th equivalence class, the family $\mathcal{F}_i$ is the projection of $\mathcal{F}_V$ onto the artifact names of the class.*

($\mathcal{C}3$) For all i, $1 \leq i \leq n$, $\mathcal{F}_{i,j}$ are the k_i equivalence classes of $\mathcal{F}_i$ w.r.t. implementation sharing $N^*_{\mathcal{F}_i}$.

The decomposition into canonical form is clearly unique for a simple family, and exposes one level of hierarchy. Thus, by iterative application of the decomposition, we obtain a mapping from families to hierarchical variability models.

Definition 10 (Variability model generation). *The mapping $shvm(\mathcal{F})$ from simple families presented in canonical form to variability models is inductively defined as follows:*

$$shvm(\{M_C\}) \stackrel{\mathsf{def}}{=} M_C$$

$$shvm\big(\{M_C\} \bowtie \textstyle\prod_{1\leq i\leq n} \bigcup_{1\leq j\leq k_i} \mathcal{F}_{i,j}\big) \stackrel{\mathsf{def}}{=} (M_C, \{VP_1, \ldots, VP_n\})$$

$$\text{where } VP_i \stackrel{\mathsf{def}}{=} \{shvm(\mathcal{F}_{i,j}) \,|\, 1 \leq j \leq k_i\}$$

We say that family $\mathcal{F}$ generates *variability model $shvm(\mathcal{F})$.*

As the next result shows, the generated variability model is well-formed.

Proposition 3. *If family $\mathcal{F}$ is simple, then $shvm(\mathcal{F})$ is well-formed.*

Example 7. Consider the family $\mathcal{F}_A$ from Example 1.

- In the first step of the decomposition of $\mathcal{F}_A$ into canonical form we obtain the common set $P = \{a_1, b_1\}$ and the family $\mathcal{F}_V = \{\{c_1, d_1, e_1\}, \{c_1, d_1, e_2\}, \{c_2, d_2, e_1\}, \{c_2, d_2, e_2\}, \{c_2, d_3, e_1\}, \{c_2, d_3, e_2\}\}$.
- In the next step, we analyze $\mathcal{F}_V$ to find that only artifact names c and d are correlated. Projecting $\mathcal{F}_V$ onto the two resulting equivalence classes $\{c, d\}$ and $\{e\}$ we obtain the two variation points $\mathcal{F}_1 = \{\{c_1, d_1\}, \{c_2, d_2\}, \{c_2, d_3\}\}$ and $\mathcal{F}_2 = \{\{e_1\}, \{e_2\}\}$.
- In the third step, we analyze $\mathcal{F}_1$ and see that two products share the artifact implementation c_2, which gives us the variants $\mathcal{F}_{1,1} = \{\{c_1, d_1\}\}$ and $\mathcal{F}_{1,2} = \{\{c_2, d_2\}, \{c_2, d_3\}\}$, and then analyze $\mathcal{F}_2$ to obtain the variants $\mathcal{F}_{2,1} = \{\{e_1\}\}$ and $\mathcal{F}_{2,2} = \{\{e_2\}\}$.

Only $\mathcal{F}_{1,2}$ is not a ground model. Applying the above steps decomposes it into a common set $\{c_2\}$ and a single variation point with two variants consisting of the common sets $\{d_2\}$ and $\{d_3\}$. It is easy to see that $shvm(\mathcal{F}_A)$ is the variability model $\mathcal{S}_{A2}$ in Figure 2.

3.3 Characterization Results

Our first result establishes *correctness* of model extraction.

Lemma 1. *For every simple family $\mathcal{F}$ we have:*

$$family(shvm(\mathcal{F})) = \mathcal{F}$$

The second result establishes *uniqueness* of well-formed models w.r.t. the generated (simple) family.

Lemma 2. *For every well-formed variability model $\mathcal{S}$ we have:*

$$shvm(family(\mathcal{S})) = \mathcal{S}$$

An immediate consequence of the above two lemmata is our main characterization result, which essentially states that the two transformations relating variability models and families are inverses of each other.

Theorem 1 (Characterization Theorem). *For every simple family $\mathcal{F}$ and every well-formed variability model $\mathcal{S}$ we have:*

$$family(\mathcal{S}) = \mathcal{F} \iff shvm(\mathcal{F}) = \mathcal{S}$$

4 Application

In this section, we show how to apply our theory to families consisting of products of program code. We explain how to obtain an SHVM from a set of products, and what insights one can gain from the derived model. Our running example (Section 4.1) is a simple product family written in Java, but the application of our theory is not restricted to particular programming languages or paradigms.

4.1 Example Product Line: Storing and Processing Collections

The example family consists of six products, where each product is a Java class. The code for all products appears in Figure 4.[1] The six products—named $P_{\mathtt{X1}}$, $P_{\mathtt{X2}}$, $P_{\mathtt{Y1}}$, $P_{\mathtt{Y2}}$, $P_{\mathtt{Z1}}$, and $P_{\mathtt{Z2}}$ after the respective class—have the following commonalities: They all store a collection of values of the custom type `Elem`, have a method for setting this state to some value, a method `process()`, and last a method `compute()` which returns some subclass of `Number`. The products have the following differences: The type of the state is either `List` or `Set`, both subinterfaces of `java.util.Collection`. In the case of `List`, method `compute()` returns a `Double`, and in the case of `Set`, it returns either a `Byte` or an `Integer`. Furthermore, method `process()` either prints out the state of one element at a time using a method on class `System`, or it produces a `String` from the elements and returns it.

4.2 From Code to Artifacts

Before we can construct an SHVM, we need a scheme to obtain a set of products, that is, products in the sense of Definition 1. Thus, we must identify artifacts in the product code. An artifact name in the program code is a construct that may

[1] We have omitted the following: import declarations, definition of custom type `Elem`, and repeated or irrelevant code.

```
class X1 {
    List<Elem> state = new ArrayList<Elem>();
    void setState(List<Elem> arg) { this.state.addAll(arg); }
    Double compute() { ... }
    void process() { for (Elem e : state) System.out.println(e); }
}
class X2 {
    List<Elem> state = new ArrayList<Elem>();
    void setState(List<Elem> arg) { ... } // as before
    Double compute() { ... }
    String process() {
        String res = "";
        for (Elem e : state) res = res + "," + e.toString();
        return res;
}   }
class Y1 {
    Set<Elem> state = new HashSet<Elem>();
    void setState(Set<Elem> arg) { this.state.addAll(arg); }
    Byte compute() { ... }
    void process() { ... } // as before with same sig.
}
class Y2 {
    Set<Elem> state = new HashSet<Elem>();
    void setState(Set<Elem> arg) { ... } // as before
    Byte compute() { ... }
    String process() { ... } // as before with same sig.
}
class Z1 {
    Set<Elem> state = new HashSet<Elem>();
    void setState(Set<Elem> arg) { ... } // as before
    Integer compute() { ... }
    void process() { ... } // as before with same sig.
}
class Z2 {
    Set<Elem> state = new HashSet<Elem>();
    void setState(Set<Elem> arg) { ... } // as before
    Integer compute() { ... }
    String process() { ... } // as before with same sig.
}
```

Fig. 4. Example product line consisting of six Java classes

Table 1. Example scheme for obtaining artifacts from Java code

Art. name	*Art. impl.*	*Connection*	*Notation*
interface I	class C	C implements I	I_C
interface I	interface J	J subinterface of I	I_J
class C	class D	D subclass of C	C_D
type T	type T	(by convention)	T_T

occur several times, but with different realizations which are then the artifact implementations. Deciding how to identify artifacts in the code means determining what are the important parts of the code for the variability model of the product line. In general, this can be done in many ways. Here, we give one possible example.

For this example, we consider an artifact to be a pair of Java types, one being the name of the type and one being its implementation. For two Java

types to form an artifact, they must be connected as shown in Table 1. The types that form artifacts in our example are underlined in Figure 4. (Class `java.lang.Object` and interface `Collection` do not occur in the figure, but are also used.) These are some artifacts identified in the example:

- The interface `java.util.List` is connected to the interface `java.util.Collection` via the Java implements relation, giving rise to the artifact $\texttt{Collection}_{\texttt{List}}$ (omitting package prefixes).
- The Class `java.lang.String` is connected to the class `java.lang.Object` via the subclass relation, so we have the artifact $\texttt{Object}_{\texttt{String}}$.
- Class `Elem` is—by convention—related to itself, so we have the artifact $\texttt{Elem}_{\texttt{Elem}}$.

With the scheme in Table 1, we identify the following set of products which is a simple family and yields a hierarchical variability model with three variation points including one inside the other.

$$P_{\texttt{X1}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{List}}, \texttt{Number}_{\texttt{Double}}, \texttt{Object}_{\texttt{System}}\right\}$$
$$P_{\texttt{X2}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{List}}, \texttt{Number}_{\texttt{Double}}, \texttt{Object}_{\texttt{String}}\right\}$$
$$P_{\texttt{Y1}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{Set}}, \texttt{Number}_{\texttt{Byte}}, \texttt{Object}_{\texttt{System}}\right\}$$
$$P_{\texttt{Y2}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{Set}}, \texttt{Number}_{\texttt{Byte}}, \texttt{Object}_{\texttt{String}}\right\}$$
$$P_{\texttt{Z1}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{Set}}, \texttt{Number}_{\texttt{Integer}}, \texttt{Object}_{\texttt{System}}\right\}$$
$$P_{\texttt{Z2}} = \left\{\texttt{Elem}_{\texttt{Elem}}, \texttt{Collection}_{\texttt{Set}}, \texttt{Number}_{\texttt{Integer}}, \texttt{Object}_{\texttt{String}}\right\}$$

4.3 Constructing and Interpreting the SHVM

From the set of products obtained in the previous section, constructing an SHVM is straightforward by the procedure specified in Definition 10. We obtain the SHVM depicted in Figure 5. The SHVM in this figure is nearly identical to $\mathcal{S}_{A2}$ in Figure 2—differing only in the cardinality of set at the leftmost branch from the root. Hence, the construction proceeds similarly to that of Example 7. Since the family is simple, the obtained model is well-formed and, thus, optimal w.r.t. the separation degree.

The constructed SHVM may be read as a graphical summary of the textual product line description given in Section 4.1, focusing on Java types. Note, in particular, that the choice between `List` and `Set` is clearly visible as a variation point, and that, for example, the combination of `List` and `Byte` is not allowed by the SHVM, whereas `List` and `Double` is allowed.

5 Related Work

The existing approaches to represent solution space product line variability can be divided into three directions [23]. First, annotative approaches consider one

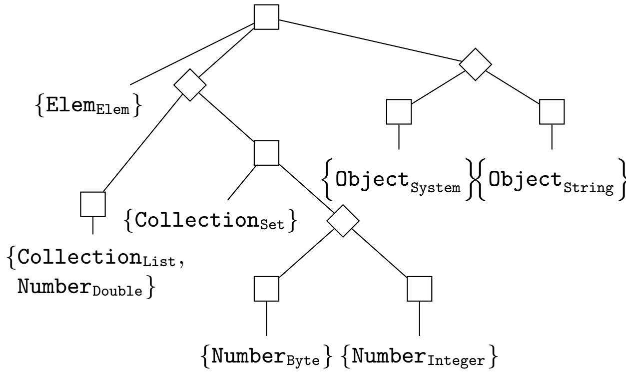


Fig. 5. SHVM for the example family

model representing all products of a product line. Variant annotations, *e.g.*, using UML stereotypes [24,10], presence conditions [6], or separate variability representations, such as orthogonal variability models [18], define which parts of the model have to be removed to generate the model of a concrete product. Second, compositional approaches [4,23,16,3] associate product fragments with product features which are composed for particular feature configurations. Third, transformational approaches [13,5] represent variability by rules determining how a base model has to be changed for a particular product model. All these approaches consider a representation of artifact variability without any hierarchy.

Our hierarchical variability model generalizes the ideas of the Koala component model [22] for the implementation of variant-rich component-based systems. In Koala, the variability of a component is described by the variability of its sub-components which can be selected by *switches* and explicit *diversity interfaces*. Diversity interfaces and switches in Koala can be understood as concrete language constructs targeted at the implementation level to express variation points and associated variants. Plastic partial components [17] are an architectural modeling approach where component variability is defined by extending partially defined components with variation points and associated variants. However, variants cannot contain variable components so this modeling approach is not truly hierarchical. Hierarchical variability modeling for software architectures [12] applies the modeling concepts for solution space variability presented in this paper to component-based software engineering and provides a concrete modeling language for variable software architectures that is truly hierarchical. However, none of these approaches formally defines the semantics of hierarchical variability models, nor reasons about their well-formedness or uniqueness.

To the best of our knowledge, this paper presents the first approach for constructing a hierarchical variability model for solution space variability from a given product family. So far, there have only been approaches to construct feature models for representing problem space variability for a given set of

products. Czarnecki *et al.* [8] re-construct a feature model from a set of sample feature combinations using data mining techniques [1]. Other approaches aim at constructing feature models from sample mappings between products and their features using formal concept analysis [9], for instance, to derive logical dependencies between code variants from pre-processor annotations [21], or to construct a feature model for function-block based systems after determining model variants by similarity [19]. Loesch and Ploedereder [14] use formal concept analysis to optimize feature models in case of product line evolution, *e.g.*, to remove unused features or to combine features that always occur together. Niu and Easterbrook [15] apply formal concept analysis to functional and non-functional product line requirements in order to construct a feature model as a more abstract representation of the requirements. Also, information retrieval techniques are applied to obtain a feature model from heterogeneous product line requirements [2]. Using hierarchical clustering, a tree structure of textually similar requirements is constructed. Requirement clusters in the leaves are more similar to each other than requirements clusters closer to the root giving rise to the structure of a feature model.

In our work, we abstract from the need to determine the different variants of the same conceptual entity by assuming fixed artifact names and corresponding artifact implementations. However, if we relax this assumption, techniques, such as similarity analysis [19] or formal concept analysis [9] could be applied to infer the relationship between different variants of the same conceptual entity, and thus make our approach applicable.

6 Conclusion

In this article, we present hierarchical solution space variability models for software product lines. We give a formal semantics of such models in terms of sets (or families) of products, where each product is a set of artifact implementations. We introduce the separation degree as a quality measure of hierarchical variability models. We identify well-formed variability models as a class of models for which the measure is maximal (and equal to one) and which are unique for the family they generate; the class of families generated by such models is the class of simple families. Furthermore, we present a transformation that constructs, from a simple family, the unique well-formed model that generates it, and prove uniqueness by showing that family generation and model construction are inverses of each other for this class of models. While maximal separation degree and uniqueness of models with maximal measure are theoretically appealing, in practice, product families might not be simple. Still, the separation degree is a useful measure for hierarchical variability models, and, as Examples 5 and 6 suggest, searching for the set of models with a maximal measure (not necessarily equal to one) for a given family is equally meaningful.

Future work will focus on the practical evaluation of the proposed method for variability model mining, considering in particular sets of (legacy code) products that have not been designed as a family from the outset. Further effort is planned

on generalizing the model with optional and multiple variant selections and with requires/excludes constraints between variants, and on adapting accordingly the model reconstruction transformation. Another generalization will deal with the more abstract domain of products over implementations only, where the names are not given in advance, but must be inferred. Additionally, the restriction that all variants associated to a variation point have to provide the same artifact names will be lifted. In order to integrate hierarchical variability models into software product line engineering, we aim at offering tool support for hierarchical variability modeling extending the approach presented in [12] and connecting variation points to product features captured by feature models.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference, pp. 207–216 (1993)
2. Alves, V., Schwanninger, C., Barbosa, L., Rashid, A., Sawyer, P., Rayson, P., Pohl, C., Rummler, A.: An exploratory study of information retrieval techniques in domain analysis. In: SPLC, pp. 67–76 (2008)
3. Apel, S., Janda, F., Trujillo, S., Kästner, C.: Model Superimposition in Software Product Lines. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 4–19. Springer, Heidelberg (2009)
4. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Trans. Software Eng. 30(6), 355–371 (2004)
5. Clarke, D., Helvensteijn, M., Schaefer, I.: Abstract delta modeling. In: GPCE. Springer (2010)
6. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
7. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
8. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: SPLC, pp. 22–31 (2008)
9. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer (1996)
10. Gomaa, H.: Designing Software Product Lines with UML. Addison Wesley (2004)
11. Gurov, D., Østvold, B.M., Schaefer, I.: A hierarchical variablility model for software product lines. Technical Report TRITA-CSC-TCS 2011:1, KTH Royal Institute of Technology, Stockholm, 26 pages (2011), `http://www.csc.kth.se/~dilian/Papers/techrep-11-1.pdf`
12. Haber, A., Rendel, H., Rumpe, B., Schaefer, I., van der Linden, F.: Hierarchical variability modeling for software architectures. In: Software Product Line Conference, SPLC 2011 (2011) (to appear)
13. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G., Svendsen, A.: Adding Standardized Variability to Domain Specific Languages. In: Software Product Line Conference (SPLC 2008), pp. 139–148. IEEE (2008)
14. Loesch, F., Ploedereder, E.: Optimization of variability in software product lines. In: SPLC, pp. 151–162 (2007)

15. Niu, N., Easterbrook, S.: Concept analysis for product line requirements. In: Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development, AOSD 2009, pp. 137–148 (2009)
16. Noda, N., Kishi, T.: Aspect-Oriented Modeling for Variability Management. In: Software Product Line Conference (SPLC 2008), pp. 213–222. IEEE (2008)
17. Pérez, J., Díaz, J., Soria, C.C., Garbajosa, J.: Plastic Partial Components: A solution to support variability in architectural components. In: WICSA/ECSA, pp. 221–230 (2009)
18. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer (2005)
19. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Automatic variation-point identification in function-block-based models. In: Proceedings of the Ninth International Conference on Generative Programming and Component Engineering, GPCE 2010, pp. 23–32. ACM, New York (2010)
20. Schaefer, I., Gurov, D., Soleimanifard, S.: Compositional Algorithmic Verification of Software Product Lines. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 184–203. Springer, Heidelberg (2011)
21. Snelting, G.: Reengineering of configurations based on mathematical concept analysis. ACM Trans. Softw. Eng. Methodol. 5, 146–189 (1996)
22. van Ommering, R.: Software reuse in product populations. IEEE Trans. Software Eng. 31(7), 537–550 (2005)
23. Völter, M., Groher, I.: Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In: Software Product Line Conference (SPLC 2007), pp. 233–242. IEEE (2007)
24. Ziadi, T., Hëlouët, L., Jézéquel, J.-M.: Towards a UML Profile for Software Product Lines. In: van der Linden, F.J. (ed.) PFE 2003. LNCS, vol. 3014, pp. 129–139. Springer, Heidelberg (2004)

Learning-Based Software Testing: A Tutorial

Karl Meinke, F. Niu, and M. Sindhu

School of Computer Science and Communication,
Royal Institute of Technology, 100-44 Stockholm, Sweden
karlm@nada.kth.se

Abstract. We present an overview of the paradigm of learning-based testing (LBT) for software systems. LBT is a fully automated method for specification-based black-box testing using computational learning principles. It applies the principle of *tests as queries*, where queries are either generated by a learning algorithm or by a model checker through use of a formal requirements specification. LBT can be applied to automate black-box testing of a variety of different software architectures including procedural and reactive systems. We describe some different testing platforms which have been designed using this paradigm and some representative evaluation results. We also compare LBT with related testing methods.

1 Introduction

Learning-based testing (LBT) is an emerging technology for *specification-based black-box testing* that encompasses the three essential steps of : (1) test case generation (TCG), (2) test execution, and (3) test verdict (the oracle step). It represents a quite sophisticated and powerful paradigm of testing that can be applied to many different types of software architecture. So far, LBT has been successfully applied to testing *procedural systems* in [25] and [27], and *reactive systems* in [30] and [28]. The basic idea of LBT is to automatically generate a large number of high-quality test cases by combining a model checking algorithm with an *efficient model inference algorithm*, and integrating these two with the system under test (SUT) in an iterative loop.

For effective testing, a variety of efficient learning principles such as *incremental learning* must be applied in order to make this technology fast and scalable to large SUTs. Our research into LBT has shown that, when suitably optimised, it has the capability to significantly outperform random testing in the speed with which it finds errors in an SUT.

In this tutorial we will present the basic principles of learning-based testing. Our tutorial is mainly oriented towards *testing researchers* and *test tool builders*. As this is a short tutorial, we will ignore the deeper theoretical issues involved in LBT, such as *convergence of learning*, *decidability of model checking* and *soundness and completeness of testing*. These issues have been considered elsewhere in the literature. The *methodology* of learning based testing, and its *integration*

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 200–219, 2012.
© Springer-Verlag Berlin Heidelberg 2012

into different software lifecycle models have not yet been investigated, and must be left for some future discussion.

The construction of LBT systems is technically demanding, since it requires expertise in both computational learning and model checking. Nevertheless, we will try to demonstrate that an investment in these techniques pays off in terms of efficient and flexible testing tools. Using three LBT testing tools that have been presented in the literature, we will illustrate how the LBT paradigm can be instantiated for testing different kinds of software systems. We will also review the literature on testing and learning, and compare the LBT approach with related approaches to software testing.

The organisation of this tutorial is as follows. In Section 2, we present an overview of the general principles of LBT. In Section 3, we discuss different computational learning techniques that have been used to optimise the effectiveness of testing. In Section 4, we provide a short survey of some practical LBT platforms and tools which have been implemented to evaluate the LBT paradigm on different types of SUT. We discuss typical performance results achieved with such tools. In Section 5, we review the literature on learning and testing, and we compare LBT with some similar approaches in the testing literature. Finally, in Section 6 we draw some conclusions and discuss the prospects for future research.

2 What Is Learning-Based Testing?

It is easier to define the paradigm of learning-based testing if we first consider the more general problem of specification-based testing.

2.1 Specification-Based Testing

In specification-based testing, we are mainly concerned with so called *functional* or *behavioural testing*, although in the case of some systems such as real-time systems, behaviour and performance cannot be entirely separated.

The starting point of specification-based testing is a user requirement $SysReq$ (a black-box requirement) concerning the functional behaviour of the system under test. Ideally such a requirement is expressed in a formal logical language such as a first-order or temporal logic. This has the significant advantage that we can then automate both test case generation, and verdict construction.

A generic architecture for specification-based testing is presented in Figure 2.1. In this figure we see that the first step in specification-based testing is to apply a constraint solving algorithm to $SysReq$. This step generates a test case for an SUT as an input datum i satisfying any preconditions or input constraints expressed in $SysReq$. Executing this test case i on the SUT using a runtime environment yields an observed output datum o. Finally, we can analyse the input/output pair (i, o) together with the requirement $SysReq$ using a constraint checker to derive a verdict about the test case i. This last phase is commonly known in testing as the *oracle step*. If the observed input/output pair (i, o) agrees with the requirement $SysReq$ then the test case is *passed*, otherwise

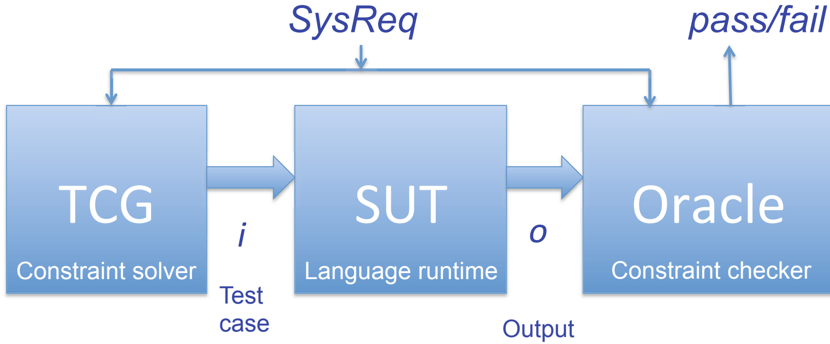


Fig. 2.1. Specification-based Test Framework

this observed black-box behaviour contradicts *SysReq* and the test case is *failed*. Note that while constraint checking is usually a decidable problem, constraint solving is often undecidable, due to the recursive insolvability of the satisfisfiability problem for many logics. Nevetheless, in some circumstances well known decision algorithms exist, and some of these are practically useful.

A small pedagogical example of these concepts is illustrated in Figure 2.2. Here we apply this generic model of specification-based testing to unit testing a simple procedural program. Generally, a unit of a procedural program takes a single input vector and returns a single output vector, and timing issues do not arise, although the unit may fail to terminate. In Figure 2.2 we are unit testing a small numerical program that implements a square root algorithm, such as Newton's method. Of course in black-box testing the structure of the code is irrelevant, and it is only the black-box behaviour which is important. Functional requirements on procedural code such as this are conventionally expressed using a *pre-* and *postcondition* or *Hoare triple*. In this example, we take the simple functional specification $\{x \geq 0\}$ SUT $\{ \; |y*y-x| \leq \varepsilon \; \}$, which expresses that for a positive input value x the absolute error of the square of the returned value y lies within a small constant positive error tolerance ε.

Let us suppose that the constraint solver returns an input value $x = 4.0$ satisfying the precondition $x \geq 0$. Suppose also that the SUT returns the output value $y = 2.0$. Then a simple calculation confirms that the assignment $(x := 4.0, \; y := 2.0)$ satisfies the postcondition $|y*y-x| \leq \varepsilon$ for any choice of positive ε.

Figure 2.2 is meant just to provide a short illustration of the principles of specification based testing. It is not the purpose of this tutorial to provide extensive information about this subject, and the reader is referred to surveys such as [36] for further details. On the other hand, this simple example already illustrates some of the difficulties of specification-based testing. One single test case would not normally be considered adequate as a test suite, even for a simple numerical SUT such as this. However, generally speaking we have no clear idea how to systematically generate many solutions to a precondition by constraint solving. (The previous example is deceptive since it is unusually simple.) This is

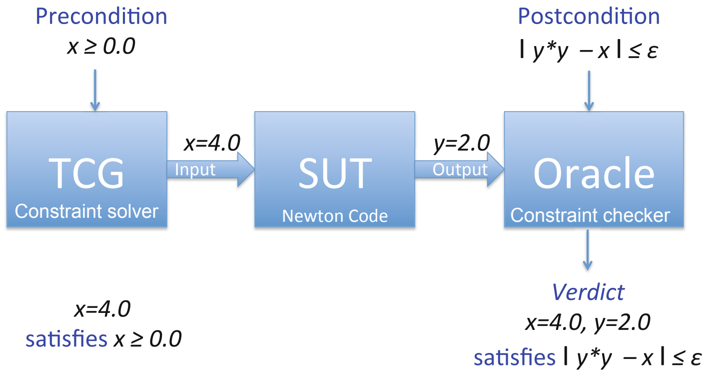


Fig. 2.2. Specification-based Testing of a procedural code unit

because it is impossible to analyse the number and distribution of solutions in the general case.

An alternative approach to the problem of generating a large number of test cases might be to introduce some sort of feedback loop into this testing process, to guide and possibly even improve the quality of subsequent test cases. Here, quality might refer to *coverage* or it might refer to *effectiveness* in finding an error. We take this possibility as our point of departure for presenting learning-based testing.

2.2 Learning-Based Testing

The goal of learning-based testing is to improve the process of specification-based black-box testing as described in Section 2.1 above. This improvement aims to automatically generate a large number of test cases within a reasonable time frame, while simultaneously optimising the quality of test cases based on the outcome of previous tests. Furthermore, the whole process should require little or no manual interaction if we are to retain the full benefits of test automation using formal requirements specifications.

As we have already suggested, the key idea is to introduce a feedback loop into the basic testing process of Figures 2.1 and 2.2. Technically, this is achieved by introducing a learning algorithm, which tries to infer a model of the unknown SUT on the basis of all currently available test data (inputs and outputs). This model of the SUT can then be automatically analysed (a process known as model checking) to try to identify counterexamples within the learned model to the correctness of the system requirements *SysReq*. Any such counterexample can then be applied as a new test case. If the model is a reasonably accurate approximation of the SUT (at least as far as the requirement is concerned) then there is a good chance that this new test case will witness a discrepancy between the observed SUT behaviour and the system requirement. In any case the accuracy of the learned model will improve over time, as new test cases are

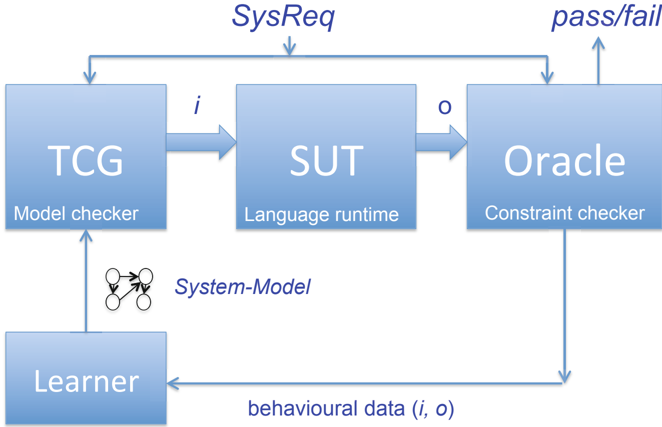


Fig. 2.3. Learning-based Test Framework

executed and integrated into it. A generic model of the LBT paradigm can be seen in Figure 2.3.

To understand this paradigm in more detail, and sketch a basic LBT algorithm, it is useful to focus on three key components:

(1) a (black-box) *system under test* (SUT) S,

(2) a *formal requirements specification* $SysReq$ for S, and

(3) a *learned model* M of S.

As we have seen, (1) and (2) are common to all specification-based testing, and it is (3) that is distinctive. Learning-based approaches are *heuristic iterative methods* to automatically generate a sequence of test cases. The heuristic approach is based on *learning a black-box system using tests as queries.*

An LBT algorithm iterates the following four steps:

(Step 1) Suppose that n test case inputs $i_1, \ldots, i_n$ have been executed on S yielding the system outputs $o_1, \ldots, o_n$. The n input/output pairs $(i_1, o_1), \ldots, (i_n, o_n)$ are synthesized into a learned model M_n of S using an efficient *learning algorithm* (see Section 3). This step involves *generalization* from the given data, (which represents an incomplete description of S) to all possible data. It gives the possibility to predict previously unseen errors in S during Step 2.

(Step 2) The system requirements $SysReq$ are satisfiability checked against the learned model M_n derived in Step 1 (aka. *model checking*). This process searches for a *counterexample* i_{n+1} to the requirements.

(Step 3) The counterexample i_{n+1} is executed as the next test case on S, and if S terminates then the output o_{n+1} is obtained. If S fails this test case (i.e. the pair (i_{n+1}, o_{n+1}) does not satisfy $SysReq$) then i_{n+1} was a *true negative* and we proceed to Step 4. Otherwise S passes the test case i_{n+1} so the model M_n was inaccurate, and i_{n+1} was a *false negative*. In this latter case, the effort of

executing S on i_{n+1} is not wasted. We return to Step 1 and apply the learning algorithm once again to $n+1$ pairs $(i_1,\ o_1),\ \ldots,\ (i_{n+1},\ o_{n+1})$ to infer a refined model M_{n+1} of S.

(Step 4) We terminate with a true negative test case (i_{n+1}, o_{n+1}) for S.

Thus an LBT algorithm iterates Steps $1 \ldots\ 3$ until an SUT error is found (Step 4) or execution is terminated. Possible criteria for termination include a bound on the maximum testing time, or a bound on the maximum number of test cases to be executed. More sophisticated models of termination can be based on the *degree of convergence* of the underlying model (c.f. [27]), which is a kind of *black-box coverage measure*.

This iterative approach to TCG yields a sequence of increasingly accurate models M_0, M_1, M_2, ..., of S. (We can take M_0 to be a minimal or even empty model.) So, with increasing values of n, it becomes more and more likely that satisfiability checking in Step 2 will produce a true negative if one exists. Notice if Step 2 does not produce any counterexamples at all then to proceed with the iteration, we must construct the next test case i_{n+1} by some other method, e.g. using a *structural query* generated by a learning algorithm, or using a *random query* (see Section 3.2). Therefore in LBT all learning can be classified as *active learning* (in the terminology of computational learning theory), since different algorithms are used to actively generate new queries during learning.

In practise, depending upon the modeling method and learning algorithm which one uses, it may not be possible to generate a new learned model M_{n+1} after each new i/o pair $(i_{n+1},\ o_{n+1})$. For example, this is true for most classical automata learning algorithms such as Angluin's L* algorithm [2], where the gap between successive hypothesis automaton constructions may be of the order of hundreds of thousands of i/o pairs. Observations of this kind have led us to investigate new learning algorithms more suitable for learning-based testing such as those of [26] and [30].

Generally speaking, we can characterise any LBT architecture in terms of:

(i) the class of *SUTs* to be tested (e.g. procedural, reactive etc.),
(ii) the class of *models* M_i,
(iii) the *learning algorithm* used to infer models M_i,
(iv) the formal *specification language* used to express requirements, and
(v) the *satisfiability algorithm* used to derive counterexamples.

Different choices from each of these categories will lead to LBT architectures with very different capability and performance properties. We will use this five-fold classification to discuss existing LBT architectures in Section 4. Note that in general it is difficult to predict testing performance from any theoretical analysis of these individual components. Therefore practical implementation and evaluation is an important aspect of research into learning-based testing.

3 Learning Principles for Efficient Testing

As has already been suggested in Section 2.2, for LBT to be *fast* at finding errors, it is important to use an efficient learning algorithm. In fact software testing places some requirements on learning algorithms which are rather novel from the traditional perspective of computational learning. Generally speaking, a good learning algorithm should maximise the opportunity of the satisfiability algorithm to find a true counterexample to the requirements *SysReq* as soon as possible. In addition, to achieve *scalability* of LBT, a number of other efficiency principles should be applied. We describe some of the most important learning principles that have been applied in this section. Further research into computational learning is still needed to extend this set of efficient techniques.

3.1 Incremental Learning

A learning algorithm L is said to be *incremental* if it can produce a sequence of models M_0, M_1, ..., known as *hypothesis models*, which are approximations to an unknown SUT S, based on a sequence of information (queries and results) about S. The sequence M_0, M_1, ... must finitely converge to S, at least up to behavioural equivalence. This principle is usually known as *learning in the limit* after the pioneering work on automata learning of Gold [18]. In addition, the computation of each new hypothesis M_{i+1} should reuse as much structural information as possible about the previous hypothesis M_i. Incremental learning algorithms can be contrasted with *complete learning algorithms*, where the emphasis is on having a sufficiently large data set (e.g. a *characteristic set of queries*) such that a model of the entire SUT S can be constructed as one single hypothesis M. By contrast, incremental learning algorithms compute a "best guess" from the currently available data.

Incremental learning algorithms are necessary for efficient learning-based testing of reactive systems for two reasons.

(1) Real systems may be too big to be completely learned and tested within a feasible timescale. This is due to their inherent complexity, and also the typical complexity properties of learning and model checking.

(2) Testing of specific requirements such as *use cases* may not require learning and analysis of an entire software system, but only of that fragment which implements the requirement *SysReq*.

Thus incremental learning improves the speed of LBT, since we can model check at more frequent intervals. It also improves the scalability of LBT, since we need not learn an entire system to find errors. In this last respect, incremental learning is similar to the technique of *program slicing* (see e.g. [23]) used in program analysis to reduce code size. Both methods can be executed dynamically. However, program slicing is a glass box technique, while LBT is a black-box technique.

To take an example: for automata, as widely used models of reactive sytems, there is an extensive literature on the problem of learning (also known as *regular inference*). A recent survey is [14]. In practise, most of the well-known classical regular inference algorithms such as L* (Angluin [2]) or ID (Angluin [1])

are designed for complete rather incremental learning. Among the much smaller number of known incremental learning algorithms, we can mention the RPNII algorithm (Dupont [15]), the IID algorithm of (Parekh et al. [32]) and IDS algorithm of (Meinke and Sindhu [29]) which learn Moore automata, the CGE algorithm (Meinke [26]) which learns Mealy automata and the IKL algorithm of (Meinke and Sindhu [30]) which learns deterministic Kripke structures.

3.2 Query Types

As we have already observed in Section 2.2, learning in LBT is *active learning.* This means that one or more algorithms are used to generate queries (as test cases) during the learning process. By considering the kinds of query generators used, and the types of queries they construct, we can influence the efficiency of testing.

For testing efficiency, we need to be aware that the overhead of SUT execution to answer a query can be large compared with the execution time of the learning algorithm itself (see e.g. [5]). So queries should be seen as "expensive", which means that we should try to choose the most efficient type at any time.

So far, based on the presentation we have given, we can identify three different types of query, namely:

(i) *model checking queries* generated by model checkers,
(ii) *structural queries* generated by learning algorithms, and
(iii) *random queries* generated by random data generators.

A model checker query is a potential counterexample to a specific user requirement *SysReq*. On the other hand, a structural query is generated by a learning algorithm to identify model structure, for example to identify a valid state space partition. A random query can be applied when the methods of model checking and learning fail to generate any new queries. In practise this can be observed to happen, albeit with low frequency. However, even a random query may be required to satisfy logical constraints, such as a precondition of a user requirement. So the algorithmic *cost* of generating these different kinds of query varies very much. Generally speaking however, random queries are computationally the easiest type to generate, while model checker queries are the most expensive to generate.

With regard to their testing efficiency, empirical evidence that we have gathered (using the LBT tools described in Section 4) suggests that: (i) random queries make the least efficient test cases, while (ii) model checking queries are more efficient than structural queries. This efficiency is measured in terms of the *number of queries* needed (on average) to find an SUT error. The *real-time performance* of learning-based testing has other factors involved as we shall see.

These efficiency differences, which can be quite significant, are mainly due to the varying relevance of each query type for falsifying a user requirement *SysReq*. Quite simply, the model checker (unlike the learner or random generator) is the only algorithm which "knows" about the user requirement. The probability of

falsifying *SysReq* on the basis of a random query is always rather low. For this reason, LBT is nearly always more efficient than random testing, unless the distribution of an SUT error is very large.

Structural queries, generated by learning algorithms, also make no reference to user requirements *SysReq*. Therefore, they too can only uncover an SUT error by accident and so perform poorly by comparison with model checker queries. However, the comparison with random queries is rather surprising. In [41] it is shown (in the context of reactive systems testing) that structural queries achieve better functional coverage than random queries. This is because most learning algorithms are optimised to discover SUT structure and converge in an efficient way, while random querying ignores any such structure.

By now we can see that the *real-time performance* of LBT involves a delicate balance between the *cost* of generating various kinds of test case versus their *effectiveness* at finding SUT errors. Model checking queries are very effective, but expensive to generate. Random queries are ineffective, but cheap to generate. These issues can be seen in practise in Section 4, when we look at the testing performance of specific LBT tools.

Let us draw some conclusions from these observations about query efficiency for the design of practical learning algorithms for LBT. We can see that when no counterexample can be found by model checking (which occurs in practise), it is generally better to use a structural query than a random query. However, it is generally better to apply a model checker query than a structural query. Therefore, as many queries as possible should be derived from model checking the hypothesis model M_i, and as few queries as possible should be derived by structural analysis of the learning algorithm. These observations have led us to investigate new learning algorithms that construct hypothesis automata using very few structural queries. For example the CGE algorithm of [26] uses a *greedy* technique that requires no structural queries at all. The RPNII learning algorithm of [15] also shares this property.

3.3 Local Learning

Instead of learning a single *global model* of an SUT, it may be possible to learn a set of smaller *local models*, which collectively model the SUT. This principle was first applied to LBT in [25], where a single global polynomial model of a numerical program was replaced by a set of piecewise overlapping polynomial models.

Where local learning can be applied, it can be shown to substantially increase the efficiency of learning and testing. On the one hand it increases the speed of learning, which usually has a time complexity that is super-linear with respect to the size of models. One can also usually restrict the number of local models that need to be updated and model checked after each new test case. Furthermore, local learning increases the speed of model checking, since again smaller local models tend to be much easier to check.

In the search for counterexamples to a user requirement *SysReq*, it may also be possible to direct the search to specific local models while ignoring others. This

choice can be made based on the estimated *reliability* of different local models (say by a *convergence analysis*), and the known absence of any counterexamples. This constitutes a novel kind of partition of the underlying SUT input space, and supports an efficient *depth first search for errors*. One example of this heuristic is described in [27], where the principle of convergence is used to estimate local model reliability and hence direct counterexample search.

3.4 System Abstraction

Abstraction has long been understood as an important principle for hiding implementation details which can make software analysis an easier task. Many examples of this principle can be applied to LBT. When used appropriately, abstraction can improve both the speed and scalability of complete LBT systems.

For example, an abstraction principle such as *data abstraction* can be applied to the choice of learned models appropriate for an SUT class (see e.g. Section 4.1 below). This choice can then make both learning and model checking much easier to carry out. Incremental learning (c.f. Section 3.1) can itself be considered as a type of abstraction technique, since those parts of an SUT which are irrelevant to a user requirement *SysReq* are ignored (abstracted away).

Other abstraction techniques such as *n-wise testing* and *bit-slicing* have also been considered in the context of LBT. (These techniques will be described later in the survey.) There are many more well-known abstraction techniques which have yet to be investigated in an LBT context.

4 A Survey of LBT Tool Architectures

The subject of learning-based testing is still in its infancy. However, already a number of complete architectures for testing different types of SUT have been designed, implemented and evaluated. In most cases, evaluation is still an ongoing activity, due to the difficulty of obtaining and testing a large number of convincing case studies. However, some useful and promising conclusions can already be made on the basis of existing tools.

Below, we describe three practical systems for learning based testing which have been documented in the literature. The first of these examples concerns testing procedural code, while the other two concern testing of embedded or reactive systems. In the context of testing reactive systems, a number of other authors, (for example Peled et al. [33], Groce et al. [19] and Raffelt et al. [37]) have also considered a combination of learning and model checking to achieve testing and/or formal verification. This work will be compared with the LBT approach in Section 5.

4.1 LBT for Procedural Systems

Architectural Summary:
SUT Type: Procedural numerical code

Model: Piecewise polynomial functions
Learning algorithm: Algebraic parameter estimation
Specification language: First-order logic over real-closed fields
Model checker: Hoon/Collins satisfiability algorithm for real-closed fields

In [25] the problem of applying LBT principles to procedural code was first considered. This resulted in a very simple LBT framework that was substantially extended in [27]. It is this latter architecture that will be discussed here.

A simple black-box model of a numerical procedural unit is a partial function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, where the real numbers are used as an idealised model or data abstraction of a specific floating point number model. We can learn such a function by splitting it into its n *co-ordinate functions* $f_1, \ldots, f_n : \mathbb{R}^m \rightarrow \mathbb{R}$. By the Weierstrass Approximation Theorem (see e.g. [35] or [38]) any continuous total function $f : \mathbb{R}^m \rightarrow \mathbb{R}$ can be approximated to arbitrary accuracy over a bounded m-dimensional sphere of radius r by an m-dimensional d-degree polynomial for some degree d. To deal with non-termination and discontinuity in numerical code, we can replace a single global polynomial (of high degree) by a set of piecewise overlapping polynomial models (of low degree) defined over several m-dimensional spheres $s_1, \ldots, s_k$ with different radii $r_1, \ldots, r_k$ and centres $c_1, \ldots, c_k$. The use of these *non-gridded centres* $c_1, \ldots, c_k$ helps reduce the exponential blowup in model size when larger values of m are involved.

Thus we bound the maximum degree d, and instead of increasing d to improve approximation, we increase the number and reduce the radii r_i of the local models instead. This gives a local modeling technique in the sense of Section 3.3. Figure 4.1 illustrates this modeling technique with a simple example.

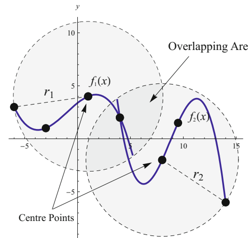


Fig. 4.1. Two cubic local models $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i,\ i = 1, 2$

Our research into LBT for procedural SUTs has focussed on numerical algorithms mainly because there is a well known and powerful algorithm for satisfiability analysis of first-order formulas over real-closed fields. This is the so called *Hoon/Collins cylindric algebraic decomposition* (CAD) algorithm (see e.g. [7]). This is a type of quantifier elimination algorithm, although practical implementations can include other methods such as Grœbner basis techniques for solving formulas. The LBT architecture of [27], uses an implementation of the CAD algorithm provided in the Mathematica$^{\text{TM}}$computer algebra package.

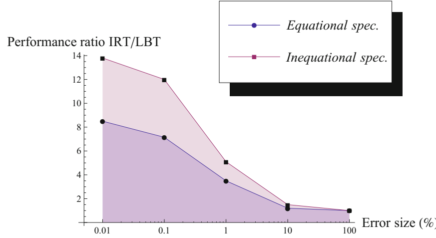


Fig. 4.2. Relative performance of iterative random testing (IRT) and LBT

Since $(d+1)^m$ points suffice to uniquely determine an m-dimensional degree d polynomial, each local model has this many points. However, local models can share points, as indicated in Figure 4.1, so that they overlap. Learning a local polynomial model with centre point c_i is easily implemented by solving a system of linear equations to estimate its coefficients, based on the $(d+1)^m$ new and previously existing data points closest to its centre point. Only those local models which are close to a new input/output pair $(i,\ o) \in \mathbb{R}^{m+1}$ need be updated. Model radii reduce naturally over time, as more new points and local models are added. Furthermore, the *relative convergence* of each local polynomial model can be estimated by integrating the volume difference between successive approximations for it. This convergence can be used as a proxy for the *reliability* of counterexamples contained within the local model. Thus if several counterexamples to *SysReq* are found by model checking then we can choose a counterexample from the most reliable local model as the next test case. This reliability approach to test case choice implements a *depth-first search* for counterexamples in the sense of Section 3.3.

When the value of m becomes large, we face a classic problem of high-dimensional modeling. This well known problem can be approached using classical methods of software testing such as n*-wise testing* (see e.g. [34]) to bound the maximum dimension of polynomial models. It has been widely observed in the literature that a large percentage of software errors can be found using low values of n. The case $n = 2$ (so called *pairwise testing*) is especially popular among testing practitioners. Thus we see in this particular LBT implementation how the basic principles of LBT can be extended with a variety of optimisations (as described in Section 3) for dealing with practical real-world testing problems.

This LBT architecture has been evaluated using both specific numerical codes and randomly generated multidimensional piecewise continuous functions. In both cases code mutations were injected to induce errors. In fact, with randomly generated functions we were able to control the *size of errors*, and investigate testing speed as a function of this size. The chosen benchmark for comparison in both cases was iterative random testing (IRT), which provides a simple and easily implemented alternative technique. One set of evaluation results presented in [27], is the relative performance graph shown in Figure 4.2. This graph demonstrates that for two different structural types of requirements formula, the

relative improvement of LBT over IRT increases steadily as the size of injected errors (and thus the ease of finding them randomly) decreases. At the limits of our data set, these speedup is on average a factor of 10-15.

4.2 LBT for Reactive Systems I

Architectural Summary:
SUT Type: Deterministic reactive systems
Model: Deterministic Boolean valued Kripke structures
Learning algorithm: IKL incremental learning algorithm for Kripke structures
Specification language: Propositional linear temporal logic
Model checker: NuSMV model checking package (SAT and BDD techniques)

A reactive system is a software system that interacts continuously with its environment over time. Typical applications include embedded software to control mechanical devices, such as automobile components, robotic machinery, flight control systems etc. Research into modeling and verifying such systems has made great progress over the last decade. It is nowadays conventional to model reactive systems as some sort of state transition system such as a *Kripke structure*, and to model user requirements on such reactive systems as temporal logic formulas. A survey of this very extensive subject is for example [12].

In [30] we introduced a new incremental learning algorithm IKL for deterministic Kripke structures. A deterministic Kripke structure can be viewed as a Moore machine, with a unique output for each state given by a Boolean vector $(o_1, \ldots, o_n) \in \mathbb{B}^n$. The IKL learning algorithm combines incremental learning with other optimisations such as *bit projection* and *lazy partition refinement* to yield an efficient algorithm for learning-based testing of reactive systems over a Boolean data type. By means of a data encoding, this approach can in principle be applied to learning reactive systems over any finite data types.

Bit projection is a technique that allows the IKL algorithm to learn just those output bits which are relevant to (i.e. appear in) a propositional temporal logic formula which formalises a user requirement *SysReq*. This abstraction technique can significantly reduce the size of the learned Kripke structure. Efficient bit projection requires on-the-fly minimisation of the state space of the learned Kripke structure. So by confining ourselves to deterministic Kripke structures we can use an efficient $n \log n$ algorithm to do minimisation. Since bit projection involves learning a single deterministic finite automaton (DFA) for each projected bit, we can apply lazy partition refinement to reduce the number of structural queries generated to learn several DFA simultaneously. This latter approach is rather technical and the reader is referred to [30] for further details.

For model checking Kripke structures we chose the NuSMV model checker (see e.g. Cimatti et al. [10]), which supports the satisfiability analysis of Kripke structures with respect to both linear temporal logic (LTL) and computation tree logic (CTL) [12]. A random query generator was also integrated to ensure that learning proceeds even when no model checker or structural queries can be generated. The resulting LBT architecture can be seen in Figure 4.3. Note in

this diagram how the oracle problem is solved by generating a *predicted output* $\overline{p}$ from the learned Kripke structure M_n and comparing this (under equality) with the *observed output* $\overline{o}$. More sophisticated oracles are possible (see Section 4.3 below), but this one is fast and works reasonably well in practise.

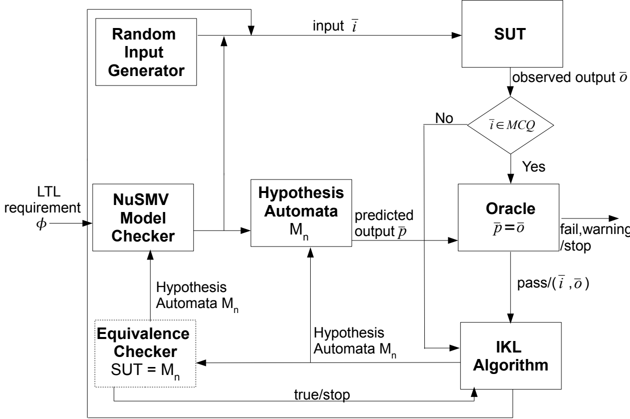


Fig. 4.3. An LBT Architecture using the IKL algorithm

Since the underlying Boolean data type allows very efficient learning and model checking (using either BDD or SAT solving techniques) the performance of this LBT architecture is quite satisfactory. Table 1 below gives the results of testing six user requirements on an elevator control program with 38 states and 8 output bits.

Table 1. LBT performance for Elevator Requirements

Requirement	t_{first} (sec)	t_{total} (sec)	% improvement ($t_{first}/t_{total} * 100$)
Req 1	0.34	1301.3	0.002
Req 2	0.49	1146	0.04
Req 3	0.94	525	0.2
Req 4	0.052	1458	0.004
Req 5	77.48	2275	3.4
Req 6	90.6	1301	7.0

We compare the average time t_{first} needed to first discover an injected error with the average time t_{total} needed to completely learn the SUT. These results show that on average incremental learning finds a first instance of an error using between 0.002% and 7% of the time needed to perform complete learning. For this specific case study and set of requirements the data supports our hypothesis about the great advantage of using incremental learning techniques.

4.3 LBT for Reactive Systems II

Architectural Summary:
SUT Type: Deterministic reactive systems
Model: Extended Mealy automata (EMA) over abstract data types
Learning algorithm: CGE incremental learning algorithm for EMA
Specification language: Quantifier-free first-order linear temporal logic
Model checker: First-order disunification algorithm using basic narrowing

Real-world reactive systems, such as flight controllers, often require high-level and complex data types that go beyond the Boolean data type. At the very least, there may be a need for *infinite data types* such as integers (counting), floating-points (measuring) and *symbolic data types* such as strings (naming). Each of these extensions results in *infinite state systems.* They may even be all combined together inside the same control algorithm which completes a complex feedback loop between sensors and mechanical actuators. Furthermore, even finite state systems, such as communication protocols may require the use of complex symbolic messages with finite parameters.

The concept of an *abstract data type* (ADT) has been introduced as a methodology for dealing with complex data models and data abstraction. The theory of *algebraic data type specification* (see e.g. [24]), provides a precise model of ADTs that can be used to design algorithms and data structures that implement this programming construct. In [26] we introduced a new symbolic learning algorithm CGE for *extended Mealy automata* (EMA). An EMA is a Mealy machine that is parameterised by algebraic data types, i.e. the input and output data for an EMA are ADTs. This combination is elegant, since automata are algebraic structures in their own right, as are ADTs. EMA provide a high-level model of reactive systems that is similar to, but simpler than, the well known Statechart model of [20]. The CGE algorithm implements symbolic learning, since the learned representation of the unknown SUT is a *string rewriting system* (SRS) over the underlying ADT. This representation can then be analysed using symbolic methods such a narrowing (see below).

Having shown that automata over high-level ADTs can be learned in principle, [28] showed how symbolic constraint solving techniques based on methods of *disunification*, [13] *term rewriting* [3] and *narrowing* [21] can be used to model check an SRS representation of an EMA. In this case the requirements language for EMA is a *quantifier-free first-order temporal logic*, since equations and inequalities concerning the underlying ADT are needed. The *basic narrowing (BN) model checker* implemented in [28] reduces the problem of model checking first-order linear temporal logic to the problem of disunification (i.e. equation solving) for sets of first-order equations and inequations by a translation method. Narrowing is a symbolic method to solve equations by means of term rewriting, while basic narrowing is a restricted form of narrowing that can be guaranteed to terminate under certain conditions. Figure 4.4 illustrates a (by now hopefully familiar) architecture for LBT that combines the CGE learning algorithm with a basic narrowing (BN) model checker.

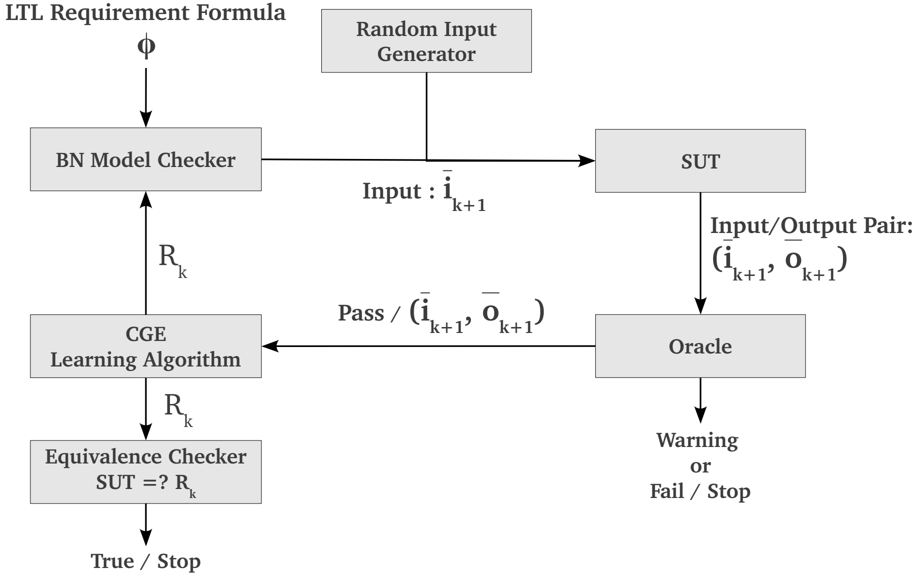


Fig. 4.4. An LBT architecture for reactive systems using symbolic learning

It is worthwhile contrasting the oracle in this architecture, with that of Section 4.2 (c.f. Figure 4.3). In the above architecture, the oracle avoids making any specific prediction of the SUT behaviour $\overline{o_{k+1}}$ that should observed on executing input $\overline{i_{k+1}}$. Instead, it uses the translation of temporal logic into first-order logic to plug the i/o pair $(\overline{i_{k+1}}, \overline{o_{k+1}})$ back into the requirement formula *SysReq* and check the truth of this formula (recall Figure 2.2 in Section 2.1). This approach to oracle construction is more flexible and powerful than the oracle described in Section 4.2. Technical details of this approach can be found in [28].

Table 2 illustrates the results of a case study using the above LBT architecture to test a simplified version of the TCP/IP protocol. This case study appeared in [28]. It involves an 11 state EMA with a symbolic input and output alphabet given by various send and receive messages with parameters. Following the evaluation principles described in Section 4.1, we again compared LBT with iterative random testing (IRT) for five different temporal logic formulas representing different user requirements. We measured the *average number of queries* Q_{first} and the *average time* t_{first} to first discover an injected error for both LBT and IRT.

The performance results of Table 2 are somewhat mixed. On the one hand, we can see that in terms of query numbers, LBT is always much more efficient than random testing. On the other hand, we can also see that the real-time performance of this LBT architecture is sometimes better and sometimes worse than iterative random testing. The essential problem here is that the overhead of learning and model checking an EMA is quite high relative to the much simpler task of randomly generating data. On the other hand, we can see that

Table 2. Random testing versus LBT: a performance comparison

Requirement	Random Testing		LBT	
	Q_{first}	$t_{first}(sec)$	Q_{first}	$t_{first}(sec)$
Req 1	101.4	0.11	19.11	0.07
Req 2	1013.2	1.16	22.41	0.19
Req 3	11334.7	36.7	29.13	0.34
Req 4	582.82	1.54	88.14	2.45
Req 5	712.27	2.12	93.14	3.13

random test data is an inefficient way to find requirements failures. Nevertheless, this data might bode well if the algorithmic methods of symbolic learning and constraint solving can be made more efficient.

The use of infinite and abstract data types in learned models is an area where much further research is still needed to make practical and efficient tools. To make matters more complicated, the satisfiability problem for many logical languages over common ADTs is undecidable. This undecidability problem afflicts both random testing, specification-based testing and LBT equally. Quite simply the search for test cases that witness a requirements failure cannot always be guaranteed to terminate.

5 Relations to Other Testing Methods

The connections between computational learning and software testing have been a fruitful line of research since the pioneering work of Budd and Angluin [6] and Weyuker's PhD research [42], in the early 1980s. Earlier approaches to test case generation by learning include work of Bergadano and Gunetti [4] on inferring and testing Prolog programs, and PAC-learning and testing of axiomatic models by Zhu and his colleagues in [44] and [43]. Complexity theoretic results on testing and learning include [9], [22] and [39].

In the context of testing reactive systems, several more recent works, (for example Peled et al. [33], Groce et al. [19] and Raffelt et al. [37]) have considered a combination of learning and model checking to achieve testing and/or formal verification. Generally, these approaches use classical algorithms for complete learning of automata such as variations of Angluin's L* algorithm. Compared with the benefits of using incremental learning algorithms, such approaches do not seem to be optimal if our aim is to use a model checker to focus on relevant test cases (c.f. the discussion of query types in Section 3.2). However, as we observed in 3.2, [41] has shown that even without model checking, test case generation by model inference alone can provide better functional coverage than random testing. So the use of complete learning algorithms is not entirely inappropriate either.

Within the model checking community the verification approach known as *counterexample guided abstraction refinement* (CEGAR) also combines learning and model checking, (see e.g. Clarke et al. [11] and Chauhan et al. [8]). The LBT

approach can be distinguished from CEGAR by both: (i) an emphasis on testing rather than verification, and (ii) optimisations on model checking and learning algorithms specifically chosen to make testing more effective..

There is an extensive literature on the emerging topic of *model-based software testing*, and the use of model checkers to automatically generate test cases. Recent surveys that focus on testing of embedded systems include [16] and [17]. Unlike LBT, these approaches involve no learning at all. Rather, they concern *glass-box testing* of reactive systems for which an automaton already exists as a design artifact or model. They generally seek to achieving specific structural coverage criteria on these models, such as *all-state* or *MC/DC*. In some sense, LBT can be seen as *model-based testing without a model* since rather than constructing it during a design process, the model is inferred from the actual implementation using test data. Thus LBT also connects to the subject of *model mining*. There are some advantages to learning a model from an SUT, not least in those situations where the design process fails to update the design model due to code changes in its implementation.

6 Conclusions and Future Research

We have presented a short tutorial on learning-based testing, and surveyed the state-of-the art in methods and tools. We have also presented a brief comparison of LBT with related testing techniques. Further discussion of both of these aspects of LBT can be found in [31] and [40], where several of the technical papers also appear.

LBT seems to be a novel, general and widely applicable paradigm. While adaptable to many different types of SUT, it also seems to have a clear set of design principles. Most importantly, it appears to be a step forward in black-box testing, from the point of view of speed at finding errors, especially those that would normally be hard to find by random testing. The high degree of automation which can be achieved in LBT also seems positive from an industrial perspective.

Implementing a successful LBT system is both challenging and rewarding. Hopefully, this tutorial is sufficient to inspire other attempts at this. However, much more research remains to be done, both in improving the efficiency and scalability of LBT, and in evaluating its performance on practical case studies. More benchmarking against other testing techniques and tools is also needed. However, the problem of finding a fair basis for comparison between different tools with different aims seems problematic. Lastly, there are still outstanding theoretical issues such as coverage measures for LBT that need to be considered.

We gratefully acknowledge financial support for this research from the Swedish Research Council (VR), the China Scholarship Council (CSC), the Higher Education Commission (HEC) of Pakistan, and the European Union under project HATS FP7-231620. We are also grateful to the anonymous referees for pointing out errors in earlier drafts of this report.

References

1. Angluin, D.: A note on the number of queries needed to identify regular languages. Information and Control 51(1), 76–87 (1981)

2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(1), 87–106 (1987)
3. Baader, F., Snyder, W.: Unification theory. In: Handbook of Automated Reasoning, pp. 447–531. Elsevier (2001)
4. Bergadano, F., Gunetti, D.: Testing by means of inductive program learning. ACM Trans. Software Engineering and Methodology 5(2), 119–145 (1996)
5. Bohlin, T., Jonsson, B.: Regular inference for communication protocol entities. Technical Report 2008-024, Dept. of Information Technology, Uppsala University (2008)
6. Budd, T.A., Angluin, D.: Two notions of correctness and their relation to testing. Acta Informatica 18, 31–45 (1982)
7. Caviness, B.F., Johnson, J.R.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer (1998)
8. Chauhan, P., Clarke, E.M., Kukula, J.H., Sapra, S., Veith, H., Wang, D.: Automated Abstraction Refinement for Model Checking Large State Spaces Using Sat Based Conflict Analysis. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD 2002. LNCS, vol. 2517, pp. 33–51. Springer, Heidelberg (2002)
9. Cherniavsky, J.C., Smith, C.H.: A recursion theoretic approach to program testing. IEEE Transactions on Software Engineering SE-13(7), 777–784 (1987)
10. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NUSMV: A New Symbolic Model Verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
11. Clarke, E., Gupta, A., Kukula, J.H., Strichman, O.: SAT Based Abstraction-Refinement Using ILP and Machine Learning Techniques. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 265–279. Springer, Heidelberg (2002)
12. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
13. Comon, H.: Disunification: a survey. In: Computational Logic: Essays in Honor of Alan Robinson, pp. 322–359. MIT Press (1991)
14. de la Higuera, C.: Grammatical Inference. Cambridge University Press (2010)
15. Dupont, P.: Incremental Regular Inference. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 222–237. Springer, Heidelberg (1996)
16. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005)
17. Fraser, G., Wotawa, F., Ammann, P.E.: Testing with model checkers: A survey. Tech. rep. 2007-p2-04, TU Graz (2007)
18. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
19. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. Logic Journal of the IGPL 14(5), 729–744 (2006)
20. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8, 231–274 (1987)
21. Hullot, J.M.: Canonical Forms and Unification. In: Proc. Fifth Int. Conf. on Automated Deduction. LNCS, vol. 87, pp. 318–334. Springer, Heidelberg (1980)
22. Vitter, J.S., Romanik, K.: Using vapnik-chervonenkis dimension to analyze the testing complexity of program, segments. Information and Computation 128(2), 87–108 (1996)
23. Krinke, J.: Program slicing. In: Handbook of Software Engineering and Knowledge Engineering. Recent Advances, vol. 3. World Scientific Publishing (2005)
24. Loeckx, J., Ehrich, H.D., Wolf, M.: Specification of Abstract Data Types. John Wiley & Sons, Inc., New York (1996)

25. Meinke, K.: Automated black-box testing of functional correctness using function approximation. In: ISSTA 2004: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 143–153. ACM, New York (2004)
26. Meinke, K.: CGE: A Sequential Learning Algorithm for Mealy Automata. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS (LNAI), vol. 6339, pp. 148–162. Springer, Heidelberg (2010)
27. Meinke, K., Niu, F.: A Learning-Based Approach to Unit Testing of Numerical Software. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 221–235. Springer, Heidelberg (2010)
28. Meinke, K., Niu, F.: Learning-Based Testing for Reactive Systems Using Term Rewriting Technology. In: Wolff, B., Zaïdi, F. (eds.) ICTSS 2011. LNCS, vol. 7019, pp. 97–114. Springer, Heidelberg (2011)
29. Meinke, K., Sindhu, M.: Correctness and performance of an incremental learning algorithm for finite automata. Technical report, School of Computer Science and Communication, Royal Institute of Technology, Stockholm (2010)
30. Meinke, K., Sindhu, M.: Incremental Learning-Based Testing for Reactive Systems. In: Gogolla, M., Wolff, B. (eds.) TAP 2011. LNCS, vol. 6706, pp. 134–151. Springer, Heidelberg (2011)
31. Niu, F.: Learning-based software testing using symbolic constraint solving methods. Licentiate thesis, School of Computer Science and Communication, Royal Institute of Technology (2011)
32. Parekh, R., Nichitiu, C., Honavar, V.G.: A Polynomial Time Incremental Algorithm for Learning DFA. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 37–49. Springer, Heidelberg (1998)
33. Peled, D., Vardi, M.Y., Yannakakis, M.: Black-box checking. In: Formal Methods for Protocol Engineering and Distributed Systems FORTE/PSTV, pp. 225–240. Kluwer (1999)
34. Phadke, M.S.: Planning efficient software tests. Crosstalk 10(10), 11–15 (1997)
35. Phillips, E.R.: An Introduction to Analysis and Integration Theory. Dover, New York (1984)
36. Poston, R.M.: Automating Specification-Based Software Testing. IEEE Computer Society Press, Los Alamitos (1997)
37. Raffelt, H., Steffen, B., Margaria, T.: Dynamic Testing Via Automata Learning. In: Yorav, K. (ed.) HVC 2007. LNCS, vol. 4899, pp. 136–152. Springer, Heidelberg (2008)
38. Reimer, M.: Multivariate Polynomial Approximation. Birkhauser, Basel (2003)
39. Romanik, K.: Approximate testing and its relationship to learning. Theoret. Comp. Sci. 188, 79–99 (1997)
40. Sindhu, M.: Incremental Learning and Testing of Reactive Systems. Licentiate thesis, School of Computer Science and Communication, Royal Institute of Technology, Stockholm, Sweden (2011)
41. Walkinshaw, N., Bogdanov, K., Derrick, J., Paris, J.: Increasing Functional Coverage by Inductive Testing: A Case Study. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 126–141. Springer, Heidelberg (2010)
42. Weyuker, E.: Assessing test data adequacy through program inference. ACM Trans. Program. Lang. Syst. 5(4), 641–655 (1983)
43. Zhu, H.: A formal interpretation of software testing as inductive inference. Journal of Software Testing, Verification and Reliability 6(1), 3–31 (1996)
44. Zhu, H., Hall, P., May, J.: Inductive inference and software testing. Journal of Software Testing, Verification and Reliability 2(2), 3–31 (1992)

Machine Learning for Automatic Classification of Web Service Interface Descriptions

Amel Bennaceur[1], Valérie Issarny[1], Richard Johansson[4],
Alessandro Moschitti[3], Daniel Sykes[1], and Romina Spalazzese[2]

[1] Inria, Paris-Rocquencourt, France
firstname.lastname@inria.fr
[2] Università degli Studi dell'Aquila, Italy
romina.spalazzese@di.univaq.it
[3] DISI, University of Trento, Italy
moschitti@disi.unitn.it
[4] Centre for Language Technology, University of Gothenburg, Sweden
richard.johansson@gu.se

Abstract. We argue that the automatic classification of web service interface descriptions into a predefined set of categories can considerably speed up the task of finding compatible web services. By doing so, we restrict computationally-expensive compatibility checking to systems within the same domain category. In this paper we show that this classification can be carried out by leveraging techniques derived from automatic document classification. In particular, we devise an approach that exploit the characteristics of web service interface descriptions to extract the features necessary for inferring the categorisation function. We further reports the results of experiments in categorising various web service interface descriptions using different classification algorithms.

1 Introduction

Interoperability is becoming more and more important given the large increase of both types and use of connecting devices, e.g., smart phones, laptop, tablets, and so on. This fact, along with the large variety of services that can be offered to the user, highly increases the communication complexity. In this perspective, automated solutions for establishing interoperability between the networked systems appear to be the only viable approach to achieve the required level of flexibility and scalability. Unfortunately, traditional solutions to determine compatibility between systems are rather expensive in terms of computational cost [4], especially when these are applied to systems in unrelated domains. Indeed, a compatibility assessment requires in-depth analyses considering the interface and conversational protocol of the two target connecting systems.

One way to speed up the assessment above is to apply machine learning methods to automatically classify high-level functionality of a system's interface description. By doing so, we restrict the scope of compatibility checks and consequently providing an overall performance gain when looking for matches between systems.

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 220–231, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In this paper, we describe how the interface description classifiers are implemented by applying machine learning: inducing the classification function from a set of examples. We describe how the standard document classification techniques need to be changed in order to be adapted to work with interface descriptions: most importantly, the feature extraction function needs to process the semi-structured data that is available in the WSDL interface description language.

We carried out a number of experiments that evaluate the effect of several design parameters in the implementation of the categorisation system, such as the design of the feature extraction function and the choice of the appropriate machine learning method.

In the reminder of the paper is structured as follows. Section 2 describes the target of the automated classifiers, i.e., the web services description files (WSDL). Section 3 describes the machine learning methods we apply, i.e., automated text categorisation, specialised for the target task. Section 4 presents our experiments on the classification of WSDL. Finally, Section 5 concludes the paper.

2 Setting Up the Context

Web services expose a description of their programmatic interface (API) using the standard WSDL (Web Service Description Language[1]) language. This description details the operations which can be performed by the web service as well as the data types of the inputs or outputs of these operations in terms of XML Schema[2]. WSDL also includes human-readable documentation and has some support for specifying the *semantics* of operations (and data) through the ontological annotations supported in SA-WSDL[3]. There is however no structured support for specifying what the service does at a high level, i.e. the abstract category to which the service belongs.

In the Connect project we define a language for specifying these categories by reference to ontology concepts [2] in order to facilitate the identification of services with similar, compatible functionality. Services in the same category are expected to have similar functionality, which may be provided to the environment, or required from it, and which allows the services to be composed and interact. However, legacy services do not make use of the explicit Connect description language. Without information about the high-level functionality of services, it is necessary to employ time-consuming syntactic and behavioural analyses that determine compatibility as a very fine-grained level of detail. Since Connect aims to overcome interoperability issues at runtime, computationally expensive procedures must be avoided. Service categories provide an interesting means to determine compatibility at the macro-level, before applying more detailed checks where necessary. Consequently we need a means to determine the high-level functionality of a service given only its WSDL description.

[1] http://www.w3.org/TR/wsdl
[2] http://www.w3.org/XML/Schema
[3] http://www.w3.org/2002/ws/sawsdl/

```
<wsdl:portType name="WeatherForecastSoap">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeSoapIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameSoapIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameSoapOut" />
  </wsdl:operation>
</wsdl:portType>
```

Fig. 1. WSDL fragment for a weather service

Figure 2 shows a partial WSDL description for a weather service, taken from the web. It lists a number of operations with names such as "GetWeatherByZipCode" and "GetWeatherByPlaceName". Each operation refers to messages which in turn determine the input and output data of the operation (defined elsewhere in the file). Each operation also includes a short piece of documentation, although this latter part is not always present. It is however obvious that this service has functionality related to the weather, and when presented with a taxonomy of categories a human would likely be able to assign this service to one of them: this is the process we seek to automate.

3 Applying Document Classification Techniques for the Categorisation of Service Descriptions

In order to build automatic classifiers of web service interfaces, we will build on the considerable amount of research that has been carried out on the topic of automatically assigning a category tag to a given text document. This is a task with many practical applications in the real world [1].

We give an introduction to the topic of automatic classification of text documents, and describe how machine learning techniques are typically applied to solve this task. We then show how these techniques can be easily adapted for the task of interface description classification. The most significant change from standard document classification methods is that the *feature extractor* – the function that determines the attributes used to distinguish the categories – needs to be tailored for the specific properties of of web service interface descriptions.

3.1 Machine Learning for Automatic Document Classification

The complexity of the category systems may vary depending on the application. The simplest would be a binary classification such as spam filtering. Slightly more complex category system can be seen in tasks such as sentiment classification of reviews [14], where the task of the classifier would be to predict the number of stars assigned by the reviewer. The largest category systems are typically hierarchically organised, such as the categories used in well-known Reuters dataset [10], and we may well imagine even more advanced categories such as the structured classifications used in library science [15].

The task of categorising documents is usually tackled by applying classifiers that have been automatically induced by estimation on a collection of documents. We refer to the process of automatically inducing a classification function from data as *machine learning*, and the collection of documents on which the estimation is carried out is called the *training set* [11].

We may divide the set of machine learning methods into two broad categories: *supervised learning*, where each document in the training set is assigned a document category by a human supervisor and the task of the machine learner is to induce a function that produces similar labelings, and *unsupervised learning*, where the documents are not labeled *a priori* and the machine learning method must find a meaningful division into categories. This paper focuses on the former method, which has generally been much more successful in most studies.

There exists a variety of methods to carry out supervised machine learning of classifiers. For classification of documents the most popular learning methods are based on the idea of associating classes of documents with regions in a *vector space*. Training a classifier becomes equivalent to describing the *decision surface*, the boundary between the regions in the space. In most cases this is described using a linear function, so the decision surface becomes a hyperplane. Methods for inducing the linear separator include the well-known perceptron algorithm [16]. The most notable recent advance in machine learning by inducing linear separator is the *support vector machine* (SVM) [3], which has been very successfully applied to the task of document classification [9]. The support vector classification approach is based on finding the maximal separation between the classes – the *maximal margin*. In case the classes are not fully separable, *soft margins* are introduced, which permit a small number of violations of the separation constraint. Figure 2 shows an illustration of a soft-margin support vector machine. Note that this is much simpler than in realistic document classification, where the documents may be represented as points in a vector space of millions of dimensions rather than just two.

In addition to the simple task of assigning a category label to a document, similar machine learning techniques can also be applied in very complex categorisation tasks. An important type of such categorisation tasks are those requiring a classification of a *pair* of documents, for instance determining whether a text is a good answer to a given question [12].

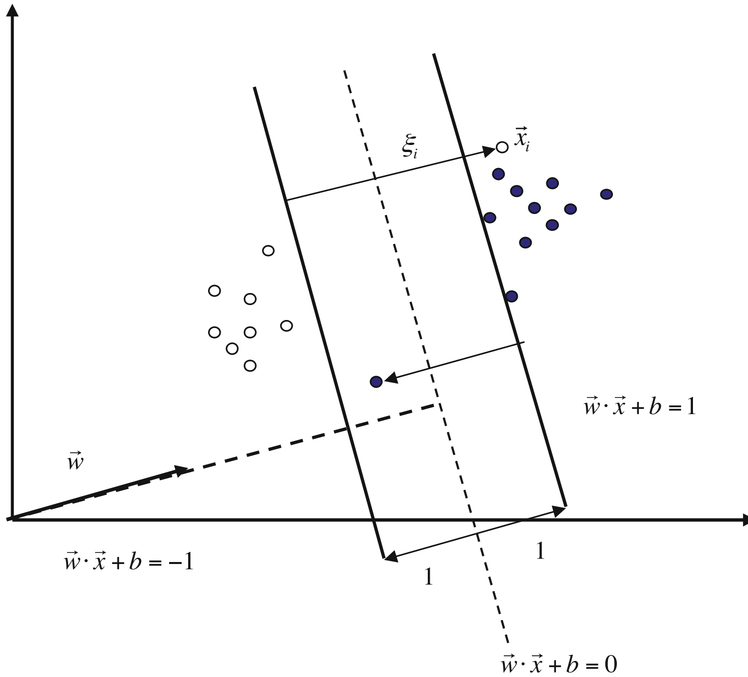


Fig. 2. Example of the decision boundary (dashed line) and the margins in a soft-margin support vector machine. There are two violations of the margin constraints.

3.2 Feature Extraction in Web Service Interface Descriptions

In order to be able to classify objects into categories, an automatic classifier needs to determine the salient properties of those objects. This process is called *feature extraction.* A good feature extractor should extract the exact distinguishing features of each category. As it has been noted repeatedly, designing feature extractors is an art more than a science, and requires a good understanding of the classification task.

For the task of document classification, the most common feature extraction method is called the *bag-of-words* representation [17]. In this method, a document is converted into a point in a vector space; every word in the vocabulary is associated with a dimension of the vector space, allowing the document to be mapped into the vector space simply by computing the occurrence frequencies of each word. The bag-of-words representation is considered the standard representation underlying most document classification approaches, and attempts to incorporate more complex structural information have mostly been unsuccessful for the task of categorisation of single documents [13].

WSDL documents generally contain a significant amount of text, and this text can be directly processed using standard a bag-of-words representation method. However, the most informative part of the WSDL interface descriptions normally

consists of *structured data*: names of methods, objects, parameters. These elements are represented as a tree-structured XML structure. It should be noted that very little textual documentation may be available if the identifiers are informatively named and their purpose obvious.

This means that the various semi-structured identifiers that are part of the WSDL interface description XML documents should be added to the bag-of-words feature representation. Most importantly, the representation should include the names of the method and input parameters defined by the interface. The inclusion of identifiers will be important since: (1) the textual content of the identifiers is often highly informative of the functionality provided by the respective methods; and (2) the free text documentation is not mandatory and may not always be present.

To extract useful word tokens from the identifiers, we split them into pieces based on the presence of underscores or CamelCase. All tokens were then normalised to lowercase. For instance, consider the following piece of WSDL code.

```
<wsdl:message name="GetWeatherByZipCodeSoapIn">
    <wsdl:part name="parameters"
          element="tns:GetWeatherByZipCode" />
</wsdl:message>
 <wsdl:message name="GetWeatherByZipCodeSoapOut">
   <wsdl:part name="parameters"
          element="tns:GetWeatherByZipCodeResponse" />
</wsdl:message>
```

In this example, we split the CamelCased identifier `GetWeatherByZipCode` into the tokens `get`, `weather`, `by`, `zip`, and `code`, so the complete bag-of-words vector for the example is `[get:4, weather:4, by:4, zip:4, code:4, soap:2, in:1, out:1, response:1]`.

4 Experiments

We carried out a large number of experiments to select the appropriate technique to implement the categorizer of web service interface descriptions.x

As described in Sec. 3.1, we implemented the classifiers by automatically inducing them from labeled data. For this purpose, we used a collection of WSDL documents[4] [8]. We selected the 10 most frequent categories, in total 397 documents as depicted in Table 1.

To train the classifiers, we used support vector machines that we trained using the LibLinear machine learning software [7].

The following subsections describe the experiments. All results have been obtained using a 10-fold cross-validation procedure: split the data into 10 pieces; form 10 different training sets by excluding each piece; train 10 classifiers; evaluate on each piece and combine the results.

[4] `http://www.andreas-hess.info/projects/annotator/ws2003.html`

Table 1. Statistics for the data collection

Category	Number of instances
CountryInfo	64
Money	54
Converter	49
Finder	46
Communication	45
Web	39
Developers	37
News	30
Business	23
Mathematics	10
Total	397

4.1 Classification Results

To evaluate the performance of the classifiers, we computed a number of different evaluation measures. The first one is the overall classification *accuracy*, which is defined as the proportion of correctly classified interface descriptions. Our best implementation correctly classified 236 out of 397 descriptions, giving us an accuracy of 59.4%.

In addition to the overall accuracy, we evaluated the classification performance on individual categories. Here, we used the *precision* (P) and *recall* (R) measures.

For a given category C, if the gold standard contains n_g instances labeled as C, our system labels n_o outputs as C, and n_c of the system outputs are correct, then we define $P = n_c/n_o$ and $R = n_c/n_g$. The overall accuracy is the number of correctly classified instances over the total number $A = \Sigma_{n_c}/\Sigma_{n_O}$ Finally, it is very common to present the harmonic mean of precision and recall, which is referred to as the F-measure.

In most situations, there is a tradeoff relationship between the precision and recall measures: if we often predict C, then we would also find many C (higher recall) but also over-generate (lower precision). By varying a class sensitivity parameter when training, we can tune the precision/recall tradeoff, and plot the relationship in a graph. Figure 3 shows such plots for the four largest classes of interfaces: CountryInfo, Money, Converter and Finder. In this kind of plot, overall prediction quality for a class is determined by how close the plot is to the upper right corner. In our case, we see that the overall prediction quality for the four classes tends to be correlated with the size of the class: CountryInfo class is the class for which the plot is closest to the upper right corner.

In addition to the class-wise precision and recall evaluations, we computed the *macro precision and recall*, which are precision and recall values averaged over all classes. In this macro evaluation, our classifier achieved a precision of 58.0, a recall of 52.8, and an F-measure of 55.3.

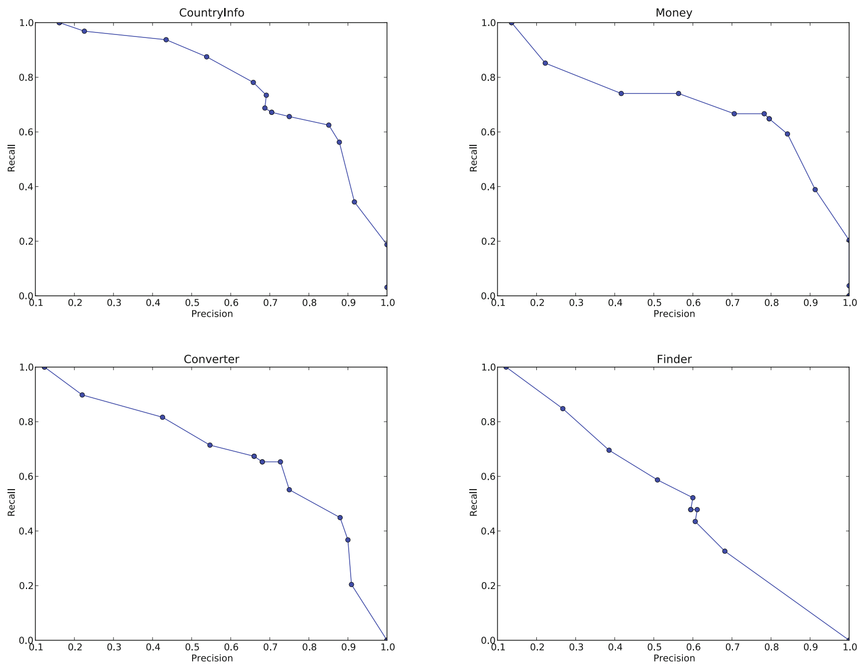


Fig. 3. Precision / recall plots for the four most frequent classes: CountryInfo, Money, Converter and Finder

4.2 Design of the Feature Extractor

The challenge in interface description classification compared to traditional document classification is the feature design problem, and in particular the problem of making use of the WSDL structure.

As a baseline, we used traditional bag-of-word feature extractors that are normally used in document classification. We applied the baseline feature extractors to the text available in the documents: the code documentation and the comments.

As an alternative to the raw text, we extracted features from the structured text, i.e. the identifiers used in the WSDL code. As discussed in Sec. 3.2, we used an identifier splitting heuristic based on the presence of CamelCase. However, we also tried out an identifier-based feature representation that did not split the identifiers.

Finally, we used a feature representation that combined the BOW and WSDL identifier features. Here, we evaluated two different representations. In the first approach we used two separate vector spaces for the two types of features. In the second one, there was only one vector space, so for instance if the word *withdraw* would appear in the documentation or in an identifier, this would result in the same feature being enabled.

To see the effect of the feature extraction design choices, we carried out an evaluation of several different feature extractors. The result of this experiment is shown in Table 2. We show the overall classification performance using accuracy and macro precision/recall/F-measure.

Table 2. Evaluation of feature extraction methods

Representation	Accuracy	Macro precision	Macro recall	Macro F-measure
BOW (doc. only)	24.9	39.5	28.4	33.0
BOW (doc. and comments)	25.2	36.1	27.8	31.5
WSDL Identifiers	41.6	33.7	32.8	33.2
WSDL Identifiers, CC splitting	58.2	54.8	51.6	53.2
BOW + Indentifiers (separate)	57.9	55.1	51.1	53.0
BOW + Indentifiers (mixed)	59.4	58.0	52.8	55.3

The experiment shows very clearly that purely text-based feature representations are not sufficient for achieving a good classification performance. It is crucial to extract features from the WSDL identifiers, and they need to be split in order to be useful. The best representation was the mixed combination of BOW and identifier features; the combination by separate vector spaces seems to lead to feature sparsity, which is very problematic when using small training sets.

4.3 Selection of the Machine Learning Algorithm

In addition to the feature extraction mechanism, another crucial parameter when building a classifier is the selection of a machine learning algorithm for training. We evaluated a wide range of learning algorithms. Our primary approach was support vector machines (SVMs) since this algorithm has been very successful in text categorization [9].

We used several variations of the SVM learning algorithm. The original SVM formulation [3] was restricted to two-class classification. This means that if we want to apply SVMs to classification problems with more than two classes, we must apply a tranformation of the problem – a *binarization*. The most common binarization method is called *one-versus-all*: create one SVM for each class, and select the highest-scoring class at test time. We tried two binarised SVM variants: L_2-loss and L_1-loss SVMs; L_2-loss is the standard formulation and L_1-loss is a newer variant that typically uses much smaller feature sets [18]. In addition to the binarised SVM variants, we used a recent SVM formulation that can solve multiclass problems directly [6].

Apart from the SVMs, we evaluated classifiers trained using the logistic regression, perceptron [16], and passive–aggressive [5] learning methods. All SVMs,

as well as the logistic regression classifiers, were implemented using LibLinear. The perceptron and passive–aggressive classifiers were based on our own implementations.

Table 3. Evaluation of machine learning methods

Learning method	Accuracy	Macro precision	Macro recall	Macro F-measure
Binarized L2-SVM	59.4	58.0	52.8	55.3
Binarized L1-SVM	58.9	55.3	49.5	52.3
Multiclass SVM	59.4	55.0	52.9	53.9
Logistic regression	49.9	47.1	39.3	42.9
Perceptron	51.1	51.0	45.6	48.1
Passive–aggressive	58.4	53.3	51.4	52.4

Table 3 shows the result of the machine learning method comparison. We notice that all SVM variants outperform the other learning algorithms. The best-performing SVM is the most commonly used variant: the L_2-loss SVM with standard binarisation.

4.4 Generating Multiple Hypotheses

The purpose of a categorizer of web services is to reduce the time it takes to decide whether we can automatically connect two different web services. While we would certainly like to have a classifier that perfectly predicts the correct class, this is of course not realistic; however, if a classifier that is allowed to make multiple predictions, the probability of the correct class being found is much higher. Such a classifier would also save considerable connection time.

We measured the performance of a classifier that is allowed to output k different category labels. In this evaluation, a prediction is counted as correct if the set of guesses contains the correct answer. The relationship between the k value and the performance is shown in Table 4. We see that even with a small k, we can get very high classification performance.

Table 4. Classification performance when predicting k possible hypotheses

k	Accuracy	Macro precision	Macro recall	Macro F-measure
1	59.4	58.0	52.8	55.3
2	71.5	71.0	65.6	68.2
3	79.3	79.1	75.0	77.0
4	85.1	85.1	81.3	83.2
5	87.7	88.0	83.9	85.9

5 Conclusion

There is a considerable need for automatic composition of web services. The CONNECT project addresses, in particular, interoperability issues arising from the need to compose heterogeneous systems at runtime. The first step in composing such systems is determining whether, and to what degree, the systems are compatible. At the highest level of abstraction, systems in the same domain category, e.g. weather, have the potential to interact.

Automatic classification of web service interface descriptions is a technique that can speed up the service matching procedure considerably by allowing us to avoid expensive behavioural analyses that encumber the runtime composition of services.

We described how methods derived from automatic document classification based on machine learning can be used to build categorizers of web service interface descriptions.

We carried out a number of experiments in automatic categorisation of interface descriptions, to determine the best way to implement an automatic system to carry out this kind of categorisation. We evaluated the effect of the choice of machine learning method, and we saw that support vector machines gave the best performance.

Most importantly, we evaluated how the performance is influenced by the design of the feature extraction component of the classifier. We saw very clearly that standard document classification methods are not directly applicable: such an approach leads to very low performance. Instead, we need to use a feature representation that is tailored to the task of interface description classification by using the specific structure of the WSDL code, in particular its identifiers.

We saw that a classifier that predicts a number of possible alternatives (not just one) achieves very high performance levels. We believe that an approach using multiple hypothesis can also be useful for service matching.

Acknowledgements. This research has been supported by the EU FP7 projects: CONNECT – Emergent Connectors for Eternal Software Intensive Networking Systems (project number FP7 231167), EternalS – “Trustworthy Eternal Systems via Evolving Software, Data and Knowledge” (project number FP7 247758) and by the EC Project, LIMOSINE – Linguistically Motivated Semantic aggregation engiNes (project number FP7 288024).

References

1. Basili, R., Moschitti, A.: Automatic Text Categorization: from Information Retrieval to Support Vector Learning. Aracne editrice, Rome (2005)
2. Bennaceur, A., Blair, G.S., Chauvel, F., Georgantas, N., Grace, P., Nundloll, V., Paolucci, M., Saadi, R., Sykes, D.: Intermediate connect architecture. Technical Report D1.2, Connect ICT FET IP Project (February 2011)

3. Boser, B.E., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Computational Learning Theory, Pittsburgh, United States, pp. 144–152 (1992)
4. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Comm. (1990)
5. Crammer, K., Dekel, O., Keshet, J., Shalev-Schwartz, S., Singer, Y.: Online passive-aggressive algorithms. Journal of Machine Learning Research 7, 551–585 (2006)
6. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. Journal of Machine Learning Research 2, 265–585 (2001)
7. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
8. Heß, A., Kushmerick, N.: Learning to Attach Semantic Metadata to Web Services. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 258–273. Springer, Heidelberg (2003)
9. Joachims, T.: Learning to Classify Text using Support Vector Machines. Kluwer/Springer, Boston (2002)
10. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A new benchmark collection for text categorization research. Journal of Machine Learning Research 5, 361–397 (2004)
11. Mitchell, T.M.: Machine Learning. McGraw-Hill (1997)
12. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: Proceedings of ACM 17th Conference on Information and Knowledge Management (CIKM), Napa Valley, United States (2008)
13. Moschitti, A., Basili, R.: Complex Linguistic Features for Text Classification: A Comprehensive Study. In: McDonald, S., Tait, J.I. (eds.) ECIR 2004. LNCS, vol. 2997, pp. 181–196. Springer, Heidelberg (2004)
14. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up? Sentiment classification using machine learning techniques. In: Proceedings of EMNLP (2002)
15. Ranganathan, S.R.: Colon Classification. Ess Ess Publications, Delhi (2006)
16. Rosenblatt, F.: The Perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65(5), 386–408 (1958)
17. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Technical Report TR74-218, Department of Computer Science, Cornell University, Ithaca, New York (1974)
18. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)

The Teachers' Crowd: The Impact of Distributed Oracles on Active Automata Learning*

Falk Howar[1], Oliver Bauer[1], Maik Merten[1],
Bernhard Steffen[1], and Tiziana Margaria[2]

[1] Technical University Dortmund, Chair for Programming Systems, Dortmund, D-44227, Germany
{falk.howar,oliver.bauer,maik.merten,steffen}@cs.tu-dortmund.de
[2] University Potsdam, Chair for Service and Software Engineering, Potsdam, D-14482, Germany
margaria@cs.uni-potsdam.de

Abstract. In this paper we address the major bottleneck of active automata learning, the typically huge number of required tests, by investigating the impact of using a distributed testing environment (a crowd of teachers) to execute test cases (membership queries) in parallel. This kind of parallelization of automata learning has the best potential when the time for test case execution is dominant, an assumption valid for most practical applications. Our investigation explicitly focuses on the impact of the structure of the system under learning (number of states, size of alphabet) and the degree of supported parallelism. It comprises three variants of active learning algorithms with different test case generation profiles. These differences can be observed directly at the level of the run-times, which all show a linear speedup for moderate degrees of parallelization, but with different saturation points beyond which further parallelization does not pay off.

1 Introduction

Automata learning techniques are becoming a valuable tool in software construction. They have been used successfully to infer models of black-box systems in the context of model-checking [16], interface synthesis [1], and (model-based) testing [6]. In all these cases, models are a necessary prerequisite, and the lack of appropriate models can therefore be considered a show stopper. Active automata learning paves the way to overcome this problem by providing a solely test-based approach for inferring models of black-box systems. Key to this approach is the active interaction with the system under learning (SUL): The heart of the automata learning process consists of an elaborate way to generate test cases tailored to support the construction of adequate system models. This directly reveals the major bottleneck of the approach: active automata learning requires the execution of enormously many test cases. Already learning models

* This work is supported by the European FP 7 project CONNECT (IST 231167).

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 232–247, 2012.
© Springer-Verlag Berlin Heidelberg 2012

with a few hundred states typically requires the execution of some hundred thousand tests case on the SUL. In classical automata learning terminology, where the considered systems are simply deterministic automata, this means that huge numbers of so-called membership queries need to be answered by a (minimally adequate) teacher also called membership oracle [2].

In this paper we investigate the impact of using a distributed testing environment (a crowd of teachers) to execute test cases (membership queries) in parallel. In our experience, this kind of parallelization of automata learning has the best potential for speedup, as it allows for quite a flexible load balancing with very little overhead. Our investigation comprises three variants of active learning algorithms with different test case generation profiles that can be observed also at the level of parallelization-based speedup.

As conceptual basis we consider the sequential L_M^* algorithm presented in [19] that can be regarded as state of the art of practical automata learning. We present here its parallel version, which essentially arises from enabling parallel execution of test cases at three dedicated points. We have implemented this version together with two similar versions for learning algorithms with slightly different querying profiles in LearnLib[1], our library for experimenting with active automata learning algorithms [17,13].

The paper presents an evaluation of these three parallel implementations in a series of experiments that explicitly focus on the impact of the structure of the SUL (number of states, size of alphabet) and the degree of supported parallelism in a context where the time for test case execution is dominant. This assumption is valid for most practical applications. The results indicate that for moderate degrees of parallelization a linear speedup can be realized, but that there seem to be saturation points beyond which further parallelization does not pay off.

Related Work. Active automata learning has been introduced in [2] for deterministic finite automata (DFAs). It has been extended to infer Mealy machine models in [15,9], which are more adequate for describing reactive systems that produce outputs rather than accept or reject sequences of inputs. It has been used successfully to infer models of black-box systems in model-checking [16], testing [6], and interface synthesis [1].

The largest reported learned model concerns a software router with 22,000 states and 7 inputs [17]. Models of actual systems are easily bigger by some orders of magnitude. Thus, models inferred by active learning are valuable assets for documenting behavior of smaller systems or for organizing test suites. This is already useful in practice even if these models are not yet used in the verification of real systems.

The performance of active learning (in terms of queries) is essential when it comes to practical application. Performance has been addressed by applying application specific filters, which help answering membership queries without performing tests on a SUL [7]. The potential of different generic types of filters and the interplay of different filters has been investigated in [11,10].

[1] `http://www.learnlib.de`

While optimizations to the number of membership queries have been investigated thoroughly, distributing queries has only been investigated from a theoretic perspective in [3]. In this paper, we investigate the impact of this optimization from a practical point of view.

Outline. We start in the next Section by providing some preliminary concepts and notation. In Section 3 we present our main result: an active automata learning algorithm that executes tests on a SUL in parallel. Section 4 presents the results of a practical evaluation of our new algorithm and compares its profile w.r.t. distribution of membership queries with some other active learning algorithms from LearnLib. Finally, we conclude in Section 5.

2 Preliminaries

In this section we define some preliminary concepts. We describe Mealy machines as a modeling formalism for a system under learning (SUL) and introduce a semantic model for Mealy machines that closely resembles regular languages. This model allows for the formulation of a Myhill/Nerode-like theorem for Mealy machines, which serves as the conceptual backbone of our learning algorithm.

Let Σ be a finite set of inputs of some system, e.g., method calls to a software library. We describe such system as an automaton.

Definition 1. *A* Mealy machine *is a tuple* $\mathcal{M} = \langle Q, q_0, \Sigma, \Omega, \delta, \lambda \rangle$ *where*

- Q *is a finite nonempty set of* states*,*
- $q_0 \in Q$ *is the* initial state*,*
- Σ *is a finite* input alphabet*,*
- Ω *is a finite* output alphabet*,*
- $\delta : Q \times \Sigma \to Q$ *is the* transition function*, and*
- $\lambda : Q \times \Sigma \to \Omega$ *is the* output function*.* □

A Mealy machine processes sequences of inputs (input words or simply words) and, other than a DFA, it produces outputs. We write $q \xrightarrow{a/o} q'$ to indicate that $\mathcal{M}$ moves from q to q' on input a producing output o according to δ and λ.

Abstracting from the fact that a Mealy machine will produce an output in every single step, we can describe the semantics of a Mealy machine as a function from the set of words to the set of outputs. Let ε denote the empty word, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ be the set of all words of length greater than zero. Then, let $[\![\mathcal{M}]\!] : \Sigma^+ \to \Omega$ be the *set of traces* of $\mathcal{M}$.

In order to describe $[\![\mathcal{M}]\!]$ in terms of $\mathcal{M}$, we inductively extend the transition function δ to $\delta^* : Q \times \Sigma^* \to Q$ by defining $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ for $q \in Q$ and $aw \in \Sigma^+$ with $w \in \Sigma^*$.

Let $[\![\mathcal{M}]\!]$ now simply be defined to be the last output of $\mathcal{M}$ on a particular word, i.e.,

$$[\![\mathcal{M}]\!](wa) \;=\; \lambda(\ \delta^*(q_0, w),\ a) \quad \text{for } wa \in \Sigma^+.$$

From this definition, we can immediately derive an equivalence relation of the set of words that resembles the Nerode relation for regular languages [14].

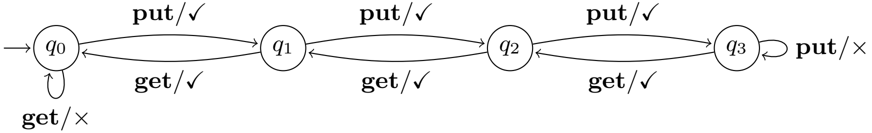


Fig. 1. Mealy machine model of a 3-place buffer

Definition 2. *For a mapping* $T : \Sigma^+ \to \Omega$*, two words* $u, u' \in \Sigma^*$ *are equivalent wrt.* T*, denoted by* $u \equiv_T u'$*, iff*

$$\forall v \in \Sigma^+ \, . \, T(uv) \; = \; T(u'v). \qquad \square$$

As for regular languages and DFAs, this relation can be used as a basis for the following characterization theorem, which we proved in [19].

Theorem 1. *A mapping* $T : \Sigma^+ \to \Omega$ *is a set of traces for some Mealy machine iff* $\equiv_T$ *has finite index.* $\square$

The theorem is a direct reformulation for Mealy machines of the well-known Myhill/Nerode theorem for DFAs. One direction of the proof of this theorem comprises the construction of the canonical Mealy machine $\mathcal{M}_T$ for T, in which every class of $\equiv_T$ corresponds to exactly one state in $\mathcal{M}_T$. This approach to automata construction is the conceptual backbone of all active automata learning algorithms. A detailed discussion can be found in [19].

Example. Figure 1 shows a graphical representation of a Mealy machine modeling a 3-place buffer with two inputs **put** and **get**, that add and remove elements from the buffer. States in the model correspond to the number of elements in the buffer. Successful actions on the buffer are indicated by output ✓ in the figure. The only two unsuccessful operations (indicated by output ×) are adding an element to the already full buffer and retrieving an element from the empty buffer. We will pick up this example when discussing the learning algorithm in Section 3. $\square$

3 Parallel Learning

In this section we present our learning algorithm that distributes test cases to multiple systems under learning SULs, and a description of how we realized the distributed implementations of the active learning algorithms within LearnLib. The results from an experimental evaluation of the gained speedup are presented in the next section.

Active learning algorithms are formulated in the MAT-learning model [2], which assumes the existence of a *Minimally Adequate Teacher* (MAT) that answers two kinds of queries.

Membership queries test for the output of a word $w \in \Sigma^+$. Sometimes $MQ(w)$ will be used instead of $[\![SUL]\!](w)$ to emphasize that the value has to be determined by a test on the SUL.

Equivalence queries test whether an intermediate hypothesis automaton $\mathcal{H}$ is equivalent to the SUL, i.e., if $[\![\mathcal{H}]\!] = [\![SUL]\!]$. If hypothesis and system are not equivalent, an equivalence query delivers a *counterexample*, i.e., a word $w \in \Sigma^+$ for which $[\![\mathcal{H}]\!](w) \neq [\![SUL]\!](w)$. Equivalence queries will be denoted by $EQ(\mathcal{H})$ in pseudo-code listings.

Corresponding to the two kinds of queries, inference is organized in two phases, alternated iteratively. In the *hypothesis construction* phase a hypothesis model is derived from the observations and iteratively refined using membership queries. In the *hypothesis validation* phase hypothesis models are tested for equivalence with the SUL by means of equivalence queries.

3.1 The L^*_M Algorithm

We present here a variant of the *reduced observation table* algorithm presented by Rivest and Schapire for determined finite acceptors (DFAs) in [18]. Our variant, the L^*_M algorithm, infers Mealy machine models and is described in [19], that also presents an extended example. Here we restrict ourselves to a brief description of the realization of the two phases discussed above in the basic algorithm and focus rather on the distribution of membership queries.

Hypothesis construction. In its basic form, active learning starts with a hypothesis automaton with only one state and refines this automaton on the basis of membership queries until a consistent state-minimal deterministic hypothesis automaton can be constructed. Key to achieving this result is the dual characterization of states that is established in Definition 2 and Theorem 1:

- by a set $U \subset \Sigma^*$ of prefixes. L^*_M constructs such a set U, containing prefixes $u \in \Sigma^*$ reaching all states of the hypothesis automaton. This characterization of states is too fine, as different words $u_1, u_2 \in U$ may lead to the same state in the target system. Hence, L^*_M maintains a second set $U_s \subseteq U$ of *access sequences* for which it is guaranteed that during learning $u_1, u_2 \in U_s$ (and $u_1 \neq u_2$) definitely lead to different states in the SUL. Technically, the relation between U and U_s is $U = U_s \cup U_s \times \Sigma$.
- by an ordered set, $V \subset \Sigma^*$, of *distinguishing sequences*. L^*_M realizes the characterization of a hypothetical state reached by some prefix u in terms of a vector $\langle r_1, \ldots, r_k \rangle$ (with $r_i \in \Omega$), characterizing the states by means of subsequent outputs, i.e., $r_i = [\![SUL]\!](u \cdot v_i)$ with $v_i \in V$. Intuitively, the vector approximates a characterization of the state wrt. to $\equiv_T$ (cf. Definition 2).

L^*_M maintains its observations in an *observation table* $\langle U, V, T \rangle$, where U is the set of prefixes, V is the set of suffixes, and $T : U \times V \to \Omega$ is the *table mapping* with $T[u, v] = [\![SUL]\!](u \cdot v)$. We define the *table row* of a prefix $u \in U$, denoted by $row(u)$, to be the vector $\langle T[u, v_1], \ldots, T[u, v_k] \rangle$ for $V = \langle v_1, \ldots, v_k \rangle$. Of course, membership queries are used to construct and maintain the table mapping.

	put	**get**
ε	✓	×
get	✓	×
put	✓	✓
put put	$\|\Sigma\|\|V\|$	
put get		

	put	**get**	**put put**
ε	✓	×	$\|U\|$
put	✓	✓	
get	✓	×	
put put	✓	✓	
put get	✓	×	

Fig. 2. Two intermediate observation tables with batches of membership queries for example from Figure 1

Example. Figure 2 displays two snapshots of the observation table that is constructed when inferring a model of the 3-place buffer of Figure 1. The rows of the table are labeled with prefixes from U, the columns are labeled with suffixes from V, and the cells correspond to the table mapping; $row(u)$ is the actual row labeled by u in a table. The set U_s corresponds to the row labels in the upper parts of the tables. □

When learning, the set U_s is initialized as $\{\varepsilon\}$ and contains only the access sequence to the initial state. The set $U \setminus U_s$ is initialized accordingly as Σ and covers all transitions originating from the initial state. The ordered set V of distinguishing suffixes is initialized as Σ and characterizes states by the output that are produced along outgoing transitions.

The L_M^* algorithm then continues by refining the hypothesis. It checks whether an automaton constructed from the observation table is *closed* under the one-step transitions, i.e., if every transition from a state of this automaton ends in a well defined state of this same automaton. Since states in a hypothesis are characterized by rows in the observation table, this is the case if for every $u \in U \setminus U_s$ there exists a $u' \in U_s$ with $row(u) = row(u')$.

In case $row(u) \neq row(u')$ for some $u \in U \setminus U_s$ and all prefixes $u' \in U_s$, i.e., if the observation table is *unclosed*, the set of access sequences U_s is extended by u and U is extended by $\{u\} \cdot \Sigma$ accordingly. This procedure is iterated until the observation table is closed and a hypothesis automaton $\mathcal{H} = \langle Q, q_0, \Sigma, \Omega, \delta, \lambda \rangle$ can be constructed from the table, where:

- for every access sequence u in U_s there is a state $q_u \in Q$,
- $q_0 = q_\varepsilon$ is the state reached by the prefix ε, and
- for every prefix $ua \in U$ with $a \in \Sigma$ there is a transition $q_u \xrightarrow{a/T[u,a]} q_{u'}$ such that $row(ua) = row(u')$.

This construction resembles closely the construction of a canonical Mealy machine from a set of traces (cf. [19]). Closedness of the observation table guarantees that $\mathcal{H}$ is well-defined.

Hypothesis verification. Once a hypothesis $\mathcal{H}$ is produced from the observation table, an equivalence query can be used to find a counterexample, i.e., a word $w \in$

Algorithm 1. Parallel L^*_M

Input: A fixed set of inputs Σ
Output: A model $\mathcal{H}$ with $[\![\mathcal{H}]\!] = [\![SUL]\!]$

```
 1: U_s := {ε}                                         ▷ Initialize observation table
 2: U := U_s ∪ Σ
 3: V := Σ
 4: for all u ∈ U, v ∈ V do                            ▷ (|Σ| + 1)|V| membership queries
 5:     T[u, v] := MQ(uv)                              ▷ performed in parallel
 6: end for
 7: loop
 8:     while ⟨U, V, T⟩ not closed do
 9:         Let u in U \ U_s s.t. ∀u' ∈ U_s : row(u) ≠ row(u')     ▷ Close table
10:         U_s := U_s ∪ {u}
11:         U := U ∪ {ua | a ∈ Σ}
12:         for all a ∈ Σ, v ∈ V do                    ▷ |Σ||V| membership queries
13:             T[ua, v] := MQ(uav)                    ▷ performed in parallel
14:         end for
15:     end while
16:     construct hypothesis H from ⟨U, V, T⟩          ▷ cf. Section 3
17:     ce := EQ(H)                                    ▷ Perform equivalence query
18:     if ce = 'OK' then                              ▷ Learned successfully?
19:         return H
20:     end if
21:     decompose ce to find suffix v                  ▷ ⌈log2(|ce|)⌉ membership queries
22:     as described in Section 3                      ▷ performed sequentially
23:     V := V ∪ {v}
24:     for all u ∈ U do                               ▷ |U| membership queries
25:         T[u, v] := MQ(uv)                          ▷ performed in parallel
26:     end for
27: end loop
```

Σ^+ for which $[\![\mathcal{H}]\!](w) \neq [\![SUL]\!](w)$. In practice, an equivalence query is usually approximated by a number of membership queries, using ideas from conformance testing (e.g., [4]). Thus, equivalence queries can also be optimized by distributing membership queries to multiple instances of a SUL. This, however, exceeds the scope of this paper. Here we assume the existence of actual equivalence queries.

In [19] we proved the following theorem about counterexamples, which is a variant of the version for DFAs from [18].

Theorem 2. *A counterexample w has a suffix v such that for two prefixes $u \in U$ and $u' \in U_s$ with $u \neq u'$ and $row(u) = row(u')$ it holds that $[\![SUL]\!](u \cdot v) \neq [\![SUL]\!](u' \cdot v)$.* □

Intuitively, the theorem states that there is at least one suffix of the counterexample that will lead to unclosedness in the observation table if added to the set of distinguishing suffixes. Such a suffix can be found using a binary search on the counterexample [18,19].

Adding the suffix to the set V leads to a refined hypothesis automaton in the next equivalence query, with at least one new state. Since, on the other hand, the characterization of states by rows in the table will never refine $\equiv_T$ (which would correspond to using Σ^+ as set of suffixes), the process is guaranteed to terminate with the canonical Mealy machine for SUL.

Put together, this results in Algorithm 1, where lines 1-6 initialize the observation table and lines 8-15 close the observation table as described above. In lines 16-17 an equivalence is used to search for a counterexample. In case none is found, the algorithm terminates successfully with the correct model in line 19. Otherwise, a new suffix is extracted from the counterexample and added to the table in lines 21-26.

3.2 Parallel Execution of Queries

In Algorithm 1 the parts between **for all** *domain* **do** and **end for** are meant to be parallelized, where *domain* defines a set of objects to be used in the different threads of execution. In particular, the algorithm performs *batches* of membership queries in parallel

- when initializing the observation table (lines 4-6),
- when closing the observation table (lines 12-14), and
- when adding new suffixes from counterexamples (lines 24-26).

The membership queries used in the analysis of counterexamples (lines 21-22), on the other hand, cannot be executed in parallel since the binary search on the counterexample cannot be split into multiple independent threads of execution.

Example. Figure 2 shows operations on the observation table for our running example. In the left table, the prefix **put** is added to the set of access sequences (indicated by an arrow). The resulting extension of the table is shown below: the two extensions of **put** with **put** and **get** are added to the set of prefixes. The resulting batch of membership queries has size 4 in this particular case. In the right table, the set of suffixes is extended by the new suffix **put put**, which will eventually lead to $row(\textbf{put}) \neq row(\textbf{put put})$. The corresponding batch of membership queries has size 5. □

3.3 Implementation

We have implemented a method for distributing membership queries on top of LearnLib. The conceptual idea of our realization is shown in Figure 3: The learning algorithm produces sets of membership queries (batches), which are handed to a so-called `BatchOracle`. This oracle schedules the distribution of the queries to a fixed number of `MembershipOracle`s, each of which is connected to a separate instance of the SUL.

Since this pattern is quite universal, we were able to extend all learning algorithms in LearnLib to support `BatchOracle`s. The next section discusses our experimental results showing that the different learning algorithms have different parallelization profiles.

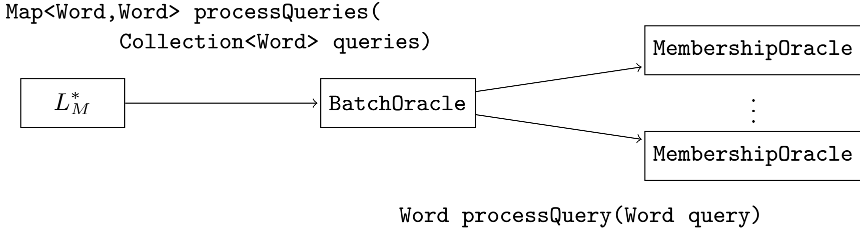


Fig. 3. Schematic overview of learning algorithm using multiple SULs in LearnLib

4 Evaluation

In this section we present the results of three series of learning experiments conducted using the newly implemented parallelized versions of the learning algorithms in LearnLib. The experiments were executed in a simulation environment that supports equivalence queries. In order to compare the algorithms w.r.t. the distribution of membership queries, we voluntarily did not use membership queries to approximate equivalence queries. For the same reason we excluded membership queries used during analysis of counterexamples from the statistics since these (few) sequentially produced membership queries occur in all algorithms. Finally, we used a cache and counted only original membership queries. In the experiments in which we measured run-times, however, the time spent for analysis of counterexamples is included in the observations.

Our evaluation focuses on the impact of the structure of the SUL (number of states, size of alphabet) and the degree of supported parallelism in a context where the time for test case execution is dominant and thus the bottleneck, an assumption valid for most practical applications. In order to model this dominance without imposing too long experimentation times, we set up the simulator to require 5 ms (realized as busy waiting) per membership query. Although 5 ms are still extremely short when considering practical applications, where several seconds are not uncommon, they were sufficient to make the time spent in processing membership queries the dominant costs during learning while still allowing to execute experiments on systems of reasonable size in an acceptable amount of time. We therefore believe that the observed speedup patterns are quite representative, and that run-times for real systems of similar size can be easily extrapolated.

In the experiments we used the parallelized versions of the following three active learning algorithms:

- $\mathbf{L^*_M}$ This is the algorithm we discussed in the previous section. It uses an observation table as underlying data structure, and it can produce batches of queries when (1) resolving an unclosedness or when (2) extending the set of suffixes as discussed in Section 3.
- **DHC** The *Direct Hypothesis Construction* (DHC) algorithm [19,12] has a different profile w.r.t. to the batches of queries produced during learning. It is

capable of resolving multiple occurrences of unclosedness at the same time, resulting in fewer but larger batches and a less uniform batch size. The implemented behavior would correspond to executing lines 8-15 of Algorithm 1 in parallel. When extending the set of suffixes the DHC algorithm behaves exactly like the L^*_M algorithm in lines 23-26.

Observation Pack Finally, the *Observation Pack* algorithm [5] uses a discrimination tree (cf. [8]) instead of an observation table. In this algorithm new prefixes are not added to the table but rather sunken into a tree. At every inner node of the tree a membership query is performed for each prefix. This leads to smaller batches than in the L^*_M algorithm: after resolving an unclosedness, k new prefixes can be sunken into the tree in parallel, resulting in batches of size at most k (the tree does not have to be balanced).

Also this algorithm uses new suffixes from counterexamples only to split one leaf of the tree (containing the set of prefixes corresponding to the incoming transitions of the hypothetical state represented at that particular leaf.) Thus, also batches produced by new suffixes tend to be smaller than in the L^*_M and the DHC algorithm.

Additionally, all three algorithms differ in the absolute number of membership queries and equivalence queries consumed during learning: While the Observation Pack algorithm uses dramatically less membership queries than the other two algorithms it uses, on the other hand, many more equivalence queries.

Experimental Setup. We conducted three series of experiments, targeted at determining the profile of these algorithms along different dimensions of variability. All the experiments were conducted using LearnLib in Potsdam, on the Automata Learning server, a 2.4GHz AMD Opteron processor with 16 cores (= 16 threads) and 256GB memory running Linux. The first two series address profile trait inherent to the algorithm-specific organization of the learning process (in Sect. 4.1) and to the behaviour's scalability on a family of examples (in Sect. 4.2) and were performed in sequential learning mode. On the contrary the third series (in Sect. 4.3) concerns speedup and makes use of the multicore capability of the server.

4.1 Different Profiles by Example

In this first series of experiments we compared the three algorithms on identical SULs and analyzed in detail the resulting profiles in terms of the number of membership queries, the number of equivalence queries, the number of batches, and size of batches. Figure 4 explicitly summarizes the resulting batch sizes for each of the considered algorithms for one particular (small) SUL with 15 states and 5 inputs. Considering our other experiments, the data displayed in the figure are, however, quite representative.

1. The L^*_M algorithm produces batches of the same size during a single phase of hypothesis construction. Batches produced when adding new counterexamples tend to be bigger in size (especially later in the progress) than batches

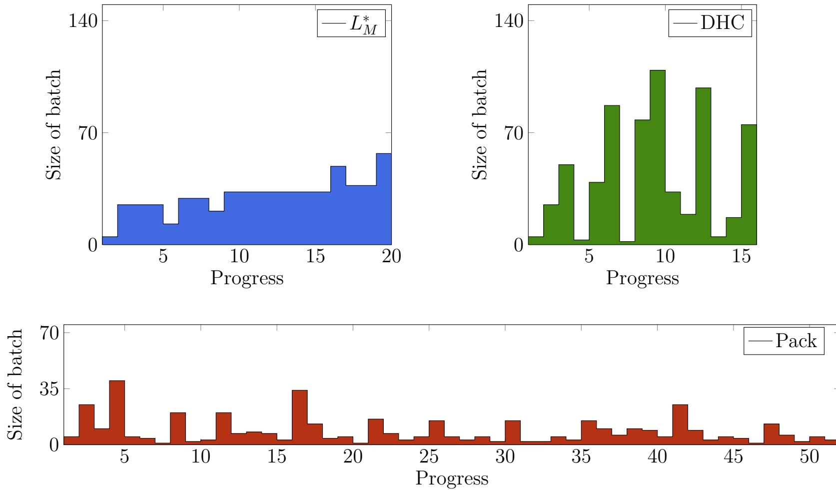


Fig. 4. Concrete numbers and sizes of batches for all algorithms and one SUL

produced by unclosedness, which grow slowly and strictly monotonically with every new suffix.

2. The DHC algorithm, on the other hand, produces fewer but large batches of queries. Each spike in Figure 4 represents one phase of model construction. In the first two batches of every spike a new suffix is added for the initial state and then to every prefix. Then, some bigger batches follow resolving multiple occurrences of unclosedness at the same time, before the phase ends much faster than in the L_M^* algorithm.
3. The Observation Pack algorithm produces significantly more and smaller batches (of sizes of ca. 5 ($= k$)) than the other algorithms. In the case of L_M^* batch sizes for most of the batches are slightly above 25 ($= k^2$).

   Finally, the Observation Pack algorithm needs less membership queries (450) than the L_M^* and DHC algorithms (624 and 650), while these need fewer equivalence queries (4 each) than the Observation Pack (10).

4.2 Asymptotic Profiles

In a second series of experiments we analyzed the mean size of batches produced by the learning algorithms for particular classes of target systems. In the first sub-series we randomly created canonical Mealy machines with 2^i states ($1 \leq i \leq 8$), 4 inputs and 8 outputs. In each class we performed 20 experiments and recorded the mean size of batches (mean of all batches of all experiments in a class).

The results are shown on the left of Figure 5. Please note that in this diagram the value axis is scaled logarithmically. The size of the batches produced by the DHC algorithm, develops almost linearly in the number of states, indicating that the number of occurrences of unclosedness that can be resolved in parallel

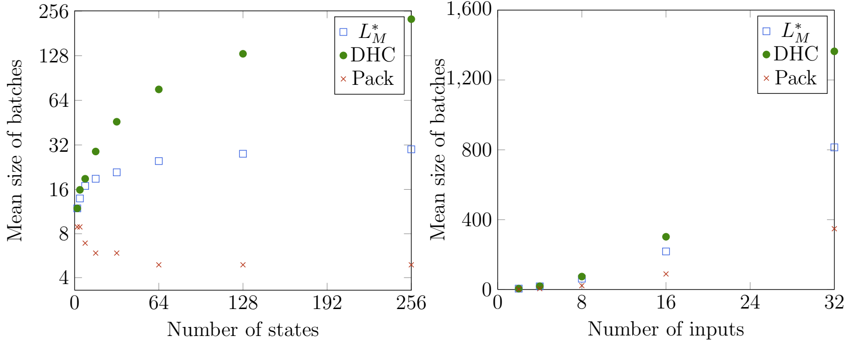


Fig. 5. Mean sizes of batches for SULs with increasing numbers of states (left) and for SULs with increasing numbers of inputs (right)

grows with the size of the inferred automata. While the mean batch sizes for the L^*_M start from k^2 and increase moderately with the number of states (and suffixes needed to distinguish these), the mean batch sizes for the Observation Pack algorithm converge towards k, as was anticipated (see above). It may look surprising at first sight that the average batch size decreases with growing system size. This is simply due to the fact that the Observation Pack algorithm starts with a batch size of k^2, and that the influence of this initial batch size becomes less relevant the longer the learning process lasts.

In the second sub-series we fixed the number of states to 10, the number of outputs to 4, and varied the number of inputs in 2^i $(1 \leq i \leq 5)$. As in the first sub-series, we performed 20 experiments in every class and recorded the mean size of batches over all batches of all experiments in a class.

Figure 5 (right) shows the mean sizes of the created batches in the experiments. As is observed easily, increasing the number of inputs has (1) a more dramatic and (2) a more uniform effect on the mean sizes of batches for all learning algorithms: The L^*_M algorithm produces batch sizes very close to k^2, which is consistent with our expectations and with the discussion above. The behavior of the other algorithms does not meet their anticipated profile. This is due to the fact that in these experiments the number of inputs clearly dominates the number of states, leading to learning processes with only very few iterations of hypothesis construction and hypothesis validation. Thus the number of membership queries issued in the initial batch has a significant influence of the mean size of all batches.

4.3 Speedup

We performed a third series of experiments in order to determine the speedup that can be gained for a fixed number (8 and 16 in this case) of SULs among which the membership queries can be distributed. For this series we have generated randomly a suite of 850 Mealy machine models, with varying number of

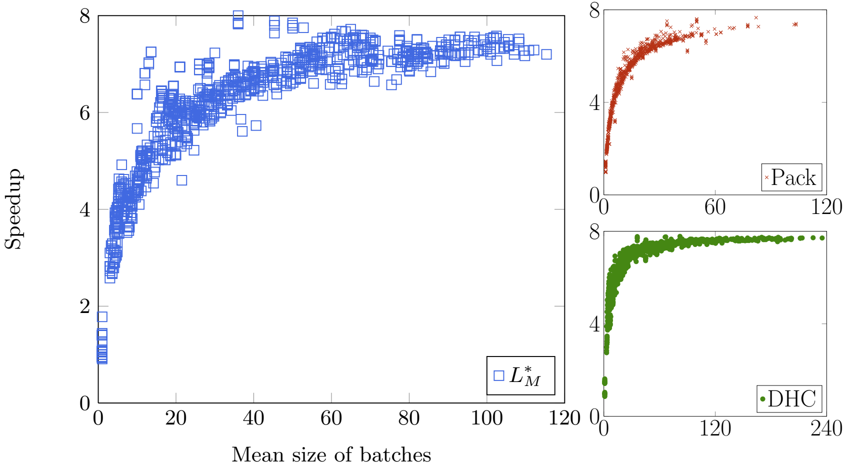


Fig. 6. Speedup per mean batch size for 8 SULs

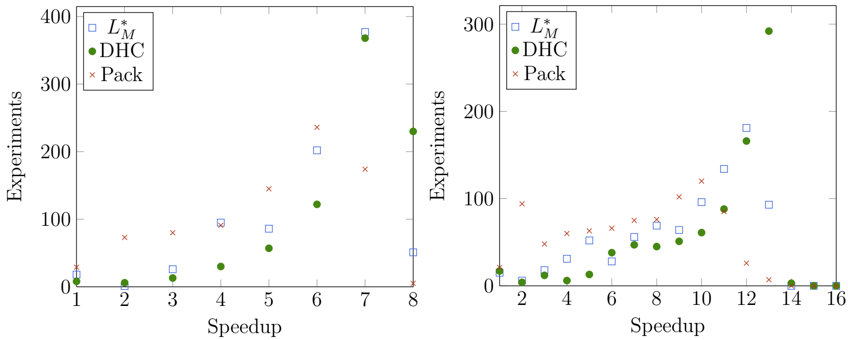


Fig. 7. Distribution of speedup in 850 experiments. Left: for 8 SULs. Right: for 16 SULs.

states, number of inputs, and number of outputs. The constructed automata had up to 10 states, up to 20 inputs, and up to 10 outputs.

For every Mealy machine in this suite, we inferred a model three times: once using only one instance of the SUL to process membership queries (used as sequential control baseline), once using 8, and once using 16 instances of the SUL to process membership queries. Each SUL was allocated to a core (thread) of the multicore server. We tried to use in different batches both the 8 and the 16 cores, in order to determine which grade of parallelism is most beneficial. As discussed above, we fixed the costs for a single membership query to 5 ms, which were spent in a busy waiting loop. The speedup in a single experiment is then the ratio between the runtime when using a single SUL and the runtime when

using 8 (16) SULs. Figure 6 shows the speedup gained in the experiments with increasing mean size of batches (per experiment this time) in the case of 8 SULs.

Considering the results, two observations are striking:

- For eight SULs a speedup of nearly 8 can be actually realized in many cases.
- A speedup of 8 is not realized for mean batch sizes of 8 (or little above) but only for mean batch sizes greater than 40.

This phenomenon has to do with the quantization effect at the number of available cores. As soon as the batch size exceeds the number of cores, the job needs to schedule an additional cycle. The last such cycle is nearly always suboptimally used, as likely some cores are not needed and remain idle. For example, using 8 SULs, the worst case is a batch of size 9: needing more than 8 cores, it will require two 5ms cycles and 10 ms to compute. In the sequential case, 9 queries would require 45 ms. This results in a speedup of just 4.5. Growing batch size reduces the impact of not using all resources for the last queries of the batch, thus reaching better speedup. For example, a batch with 41 queries would compute with a speedup of 6.8 (205ms/30ms).

The results in Figure 6 show nicely the correspondence of mean batch sizes and speedup for the learning algorithms. The results are consistent for all three algorithms. The algorithms differ, however, in the distribution of achieved speedup on the set of 850 examples. This distribution is shown in Figure 7 for the case of 8 SULs (left) and 16 SULs (right) for all algorithms. In both series of experiments the Observation Pack algorithm achieves less speedup than the L_M^* algorithm, which in turn gains less speedup than the DHC algorithm. This is little surprising since the gained speedup correlates to the mean batch sizes.

Interestingly, none of the algorithms lead to a speedup of 14 or more in the experiments with 16 SULs. This can be attributed to the fact that for these examples no algorithm produced batch sizes large enough to fully leverage 16 SULs.

Summarizing the results, one can see that the potential speedup depends hugely on the system to be inferred (i.e., its size and the size of its input alphabet). Both factors have an influence on the mean batch size. To optimally leverage distribution of queries to multiple SULs, the expected mean batch size has to be considered (a) when choosing a particular learning algorithm and (b) when determining the number of SULs to be used.

Acknowledgement. We thank Mohamed Babiker for conducting preliminary experiments.

5 Conclusion

We have investigated the impact of parallelizing the execution of test cases (membership queries) during active automata learning in a context where the time for test case execution is dominant - an assumption valid for most practical applications. Our systematic investigation has focused on the impact of the structure of

the SUL (number of states, size of alphabet) and the degree of supported parallelism. Our results indicate that for moderate degrees of parallelization a linear speedup can be realized, but that there seem to be saturation points beyond which further parallelization does not pay off.

Currently we are investigating how these results transfer to industrial-scale case studies, like e.g. the Springer's Online Conference Service (OCS), and how the different batch-profiles of the three considered algorithms show up there. Furthermore, we will investigate how this optimization can be combined with other optimizations - especially those that reduce the number of membership queries.

References

1. Alur, R., Cerný, P., Madhusudan, P., Nam, W.: Synthesis of interface specifications for Java classes. In: POPL 2005, pp. 98–109. ACM (2005)
2. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)
3. Balcázar, J.L., Díaz, J., Gavaldà, R., Watanabe, O.: An optimal parallel algorithm for learning dfa. In: Proceedings of the Seventh Annual Conference on Computational Learning Theory, COLT 1994, pp. 208–217. ACM, New York (1994)
4. Chow, T.S.: Testing Software Design Modeled by Finite-State Machines. IEEE Transactions on Software Engineering 4(3), 178–187 (1978)
5. Howar, F., Steffen, B., Merten, M.: From ZULU to RERS - Lessons Learned in the ZULU Challenge. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part I. LNCS, vol. 6415, pp. 687–704. Springer, Heidelberg (2010)
6. Hungar, H., Margaria, T., Steffen, B.: Test-based model generation for legacy systems. In: ITC 2003, pp. 971–980. IEEE Computer Society (2003)
7. Hungar, H., Niese, O., Steffen, B.: Domain-Specific Optimization in Automata Learning. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003)
8. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
9. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: HLDVT 2004, pp. 95–100. IEEE Computer Society (2004)
10. Margaria, T., Raffelt, H., Steffen, B.: Analyzing Second-Order Effects Between Optimizations for System-Level Test-Based Model Generation. In: ITC 2005. IEEE Computer Society (2005)
11. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based relevance filtering for efficient system-level test-based model generation. Innovations in Systems and Software Engineering 1(2), 147–156 (2005)
12. Merten, M., Howar, F., Steffen, B., Margaria, T.: Automata Learning with on-the-Fly Direct Hypothesis Construction. In: Hähnle, R., et al. (eds.) ISoLA 2011 Workshops. CCIS, vol. 336, pp. 248–260. Springer, Heidelberg (2012)
13. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next Generation LearnLib. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011)
14. Nerode, A.: Linear Automaton Transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)

15. Niese, O.: An Integrated Approach to Testing Complex Systems. PhD thesis, University of Dortmund, Germany (2003)
16. Peled, D., Vardi, M.Y., Yannakakis, M.: Black Box Checking. Journal of Automata, Languages and Combinatorics 7(2), 225–246 (2002)
17. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. Int. J. Softw. Tools Technol. Transf. 11(5), 393–407 (2009)
18. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Information and Computation 103(2), 299–347 (1993)
19. Steffen, B., Howar, F., Merten, M.: Introduction to Active Automata Learning from a Practical Perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)

Automata Learning with On-the-Fly Direct Hypothesis Construction*

Maik Merten[1], Falk Howar[1], Bernhard Steffen[1], and Tiziana Margaria[2]

[1] Technical University Dortmund, Chair for Programming Systems, Dortmund, D-44227, Germany
{maik.merten,falk.howar,steffen}@cs.tu-dortmund.de
[2] University Potsdam, Chair for Service and Software Engineering, Potsdam, D-14482, Germany
margaria@cs.uni-potsdam.de

Abstract. We present an active automata learning algorithm for Mealy state machines that directly constructs a state machine hypothesis according to observations, while other algorithms generate a state machine as output from information gathered in an observation table. Our DHC algorithm starts with a one-state hypothesis that it successively extends using a direct construction approach. This approach enables direct observation of the automata construction process: the learning algorithm continues to complete its hypothesis, providing intuition to a field of formal methods otherwise dominated by algorithms that largely operate on internal data structures without visible feedback.

The DHC algorithm is competitive in cases where memory is the critical issue, e.g., in embedded networked systems. It is also well-suited as educational tool to teach the underlying well-established theoretical methods in a totally unbiased fashion, without cluttering the view onto the actual idea of the learning process with aspects only relevant to internal bookkeeping.

1 Introduction

Most system documentation has central shortcomings that reduce its usefulness, sometimes to the point of being useless. Apart from missing desirable properties like being faithful to the documented system, complete, or comprehensive, it shows distinctive lack of behavioral formal models like, e.g., automata or other formats of finite state machines.

Many real-life systems can be modeled as input/output state machines, e.g., most networked applications react according to received messages and an internal state, producing state transitions and output messages in response to input messages. The same is true for hardware circuits and for many communication protocols. State machines thus are of interest for, e.g., documentation

* This work was partially supported by the European Union FET Project CONNECT: Emergent Connectors for Eternal Software Intensive Networked Systems (http://connect-forever.eu/).

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 248–260, 2012.
© Springer-Verlag Berlin Heidelberg 2012

purposes of networked systems and reactive systems in general, potentially giving insightful information on system behavior. Behavioral models provided as state machines can also be used for automated or semiautomated verification (e.g., via means of model-checking) and for simulation purposes, as the strict formalism enables automatic execution. Thus ways to (semi-)automatically derive state machine descriptions of actually deployed systems are a useful addition to the documentation-toolbox.

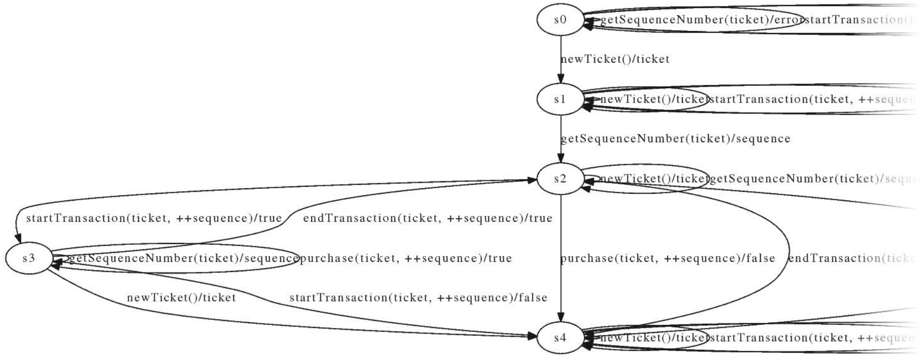


Fig. 1. Model of an exemplary web service, learned from the WSDL via automata learning

Figure 1 shows a very small example of a model created via active automata learning of a web service that simulates a vending service. Messages defined in the WSDL description can be observed, modifying the state of the service. The model reveals a tight interlock between communication primitives: e.g., the "`purchase`" primitive can only be successfully executed once a transaction was opened using the "`startTransaction`" primitive. While a WSDL does not contain such information, this is potentially valuable knowledge that any documentation for this service should contain. Automata learning can reveal such properties with little or no prior knowledge on the inner workings of exploration targets.

In this paper, we present our approach of learning automata by direct hypothesis construction (DHC). The DHC algorithm works by successively expanding a spanning tree of already explored states in a breadth first manner, and introducing cycles whenever a newly found states is considered equivalent to an already explored state until a complete hypothesis automaton has been formed. Characteristic is its notion of equivalence, which is based on the next-step pattern called *signatures*. As signatures simply based on the input alphabet are of course not enough to characterize the states of a finite state system, they are successively extended to comprise artificial symbols consisting of whole sequences of input symbols in a way mimicking the rows of the observation table underlying the well-known L^* algorithm. The result is an algorithm which continuously maintains a hypothesis and which has a significantly smaller memory footprint, an important property for our successful attempts to learn systems with more than a million states.

After providing the preliminaries in the next section, we present the DHC algorithm in Section 3, we discuss its profile in Section 4, and present our conclusions and perspectives in Section 5.

2 Scenario: Input/Output Systems

Automata learning algorithms were originally designed to learn finite state acceptors (i.e. regular languages) [2]. The approach can be extended to learn Mealy machines in a straightforward manner: only the notion of language has to be replaced by the broader notion of a semantic functional [17]. In Mealy machines input words classify w.r.t. the output symbols produced, rather than w.r.t. acceptance.

In practice Mealy machines have been shown to be useful to describe large classes of reactive systems [15,13,14,1]. Given an input symbol, a machine will produce an output symbol depending on that input and its currently active internal state, and may also let the machine switch to a another active state. The formal definition for Mealy machines is as follows:

Definition 1 (Mealy Machine)
A Mealy machine is defined as a tuple $\langle Q, q_0, \Sigma, \Omega, \delta, \lambda \rangle$ *where*

- Q *is a finite nonempty set of* states *of size* n, *i.e.:* $n = |Q|$,
- $q_0 \in Q$ *is the* initial state,
- Σ *is a finite* input alphabet *of size* k, *i.e.:* $k = |\Sigma|$,
- Ω *is a finite* output alphabet,
- $\delta : Q \times \Sigma \to Q$ *is the* transition function, *and*
- $\lambda : Q \times \Sigma \to \Omega$ *is the* output function.

Intuitively, a Mealy machine evolves through states $q \in Q$, *and whenever one applies an input symbol (or action)* $a \in \Sigma$, *the machine moves to a new state according to* $\delta(q,a)$ *and produces an output according to* $\lambda(q,a)$.
We write $q \xrightarrow{i/o} q'$ *to denote that on input symbol* i *the Mealy machine moves from state* q *to state* q' *producing output symbol* o. *The transition function* $\delta : Q \times \Sigma \to Q$ *and the output function* $\lambda : Q \times \Sigma \to \Omega$ *can be extended to process words in the obvious way.*

Learning algorithms for input/output systems use two types of queries to gather information on the target system:

- Membership Queries (MQs) retrieve behavioral information of the target system. Consisting of sequences of stimuli, MQs actively trigger the production of behavioral outputs which are collected and analyzed by the learning algorithm. MQs are used to construct a hypothesis, which after being made consistent is subject of a verification by a second class of queries, the equivalence queries.

– Equivalence Queries (EQs) are used to determine if the learned hypothesis is a faithful representation of the target system. If the equivalence oracle handling the EQ finds diverging behavior between the learned hypothesis (output function λ^{hypo}, initial state q_0^{hypo}) and the target system (output function λ^{target}, initial state q_0^{target}), a counterexample $ex \in \Sigma^*$ with

$$\lambda^{hypo}(q_0^{hypo}, ex) \neq \lambda^{target}(q_0^{target}, ex)$$

is produced, which is used to refine the hypothesis after restarting the learning process.

With those two query types learning algorithms such as $L^*_{i/o}$ [12] create minimal automata models, this meaning that the learned result never contains more states than the minimized representation of the target system, and they also guarantee termination with an accurate model.

The minimality property results from identifying states by their potential for future output behavior, which according to the well-know Myhill/Nerode Theorem [11] directly relates these states to the finitely many states of the minimized representation of the target system [17]. More concretely, the states of the learned automata correspond to the equivalence classes of the following equivalence relation between states $q_a, q_b \in Q$:

$$q_a \equiv q_b \iff \forall w \in \Sigma^+,\ \lambda(q_a, w) = \lambda(q_b, w)$$

In practice it is impossible to algorithmically consider all such futures. On the other hand, it is possible to distinguish all states of a finite state systems with finitely many futures.

Equivalence queries provide the termination property: With every counterexample the learning algorithm identifies at least one new state by using the diverging output behavior. Given that the number of states to be discovered is bound by the number of states in the target system, termination is guaranteed and the equivalence queries guarantee termination with the correct result.

3 The DHC Algorithm

Like all other active automata learning algorithms, the DHC algorithm also aims at constructing the minimal deterministic automaton of some given finite state target systems by identifying states according the to Myhill/Nerode theorem as described in the previous section. Characteristic to the DHC algorithm, however, is its consequent direct breadth-first oriented construction of intermediate hypotheses. This construction in particular avoids the use of the so-called observation tables, that may easily become a memory bottleneck.

As usual, our breadth-first search maintains a queue of states to be explored, which gets initialized with the initial state of the automaton. If a state is unique, i.e. not equivalent to any already explored state, its successors are enqueued for later exploration.

If we were able to decide the Myhill/Nerode equivalence described above this construction would immediately deliver the desired automaton. Unfortunately, in practice this equivalence can in general only be approximated on the basis of testing. The DHC algorithm therefore follows a very simple principle based on the following notion of signature, which works under the assumption that the set of input symbols is ordered:

Definition 2 (Signature). *The ordered set of output symbols produced at a state $q \in Q$ in response to applying all inputs of an ordered input alphabet is called* output signature*:* $sig(q) : (\lambda(q,a) \mid a \in \Sigma)$

Quite similarly to L^* [2] the DHC algorithm iterates now two steps:

Step 1 Construction of a (hypothesis) automaton based on the equivalence of states defined by equality of the corresponding signatures. This step always terminates with a closed and consistent hypothesis in the sense of L^*. In the following, we call states with identical signature *siblings.*

Step 2 Extension of the input alphabet in response to the counterexamples provided by the Equivalence Oracle. There are various variants of the DHC algorithms depending on the strategy of counterexample treatment. The most important ones are
- to add *all* suffixes of the counterexample to the input alphabet, which is strongly related to the counterexample treatment proposed by Maler and Pnueli [8], and
- to add only *one significant suffix* to the input alphabet according to the approaches of Rivest and Shapire [16].

These steps "aggregate" the futures provided by the counterexamples in order to be able to treat them just like individual inputs in the next iteration, with the result that the signatures directly match the state characterizing vectors of the L^* observation table.

In essence, the DHC algorithm resembles L^* very closely, with the difference that

- it continuously provides a hypothesis also during the first phase, rather than constructing it once at the end of this phase, and
- that this hypothesis, the main data structure of the DHC, is significantly smaller than observation table: it grows with $O(nk)$, whereas the observation tables grows with $O(nk^2 + n^2k)$. Unfortunately, this benefit comes with the necessity of recomputation during each iteration.

Whereas the first difference is nice for illustration or pedagogical reasons, the latter helped us in achieving our size record for active automata learning: a system with over a million states, but with an extremely fast membership oracle. The observation table for this example would have required several terabytes. Thus there was no way to fit it into main memory, which excluded this example from the scope of our algorithms.

In its first step, the DHC algorithm constructs closed and consistent hypothesis automata by sibling-based completion.

3.1 Step 1: Sibling-Based Completion

Definition 3. *A state is called* complete *if for every input symbol an output symbol and a successor state are determined. A hypothesis is called* complete *if every state of it is complete.*

Whereas it is easy to see that a complete hypothesis is closed in the sense of L^*, consistency is not so obvious. In fact, we will see that it is the result of the hypothesis construction which forces all siblings to have exactly the same successors, namely the one determined by the oldest sibling. This is in fact similar to the approach by Rivest and Shapire [16] or Maler and Pnueli [8].

The DHC algorithm is a classical worklist algorithm. It is initialized with the input-alphabet and it starts with a one-state hypothesis containing only the initial state. This hypothesis is obviously not complete, as it has no transitions yet. The initial state is therefore enqueued into the list of incomplete states, which need to be completed via additional information extracted from membership-queries.

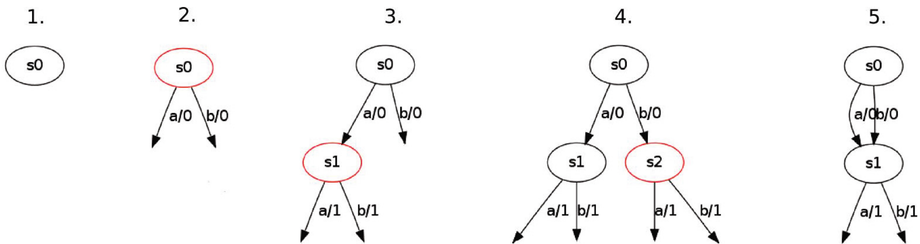


Fig. 2. First steps of DHC hypothesis construction with the alphabet {a,b}

To complete the enqueued incomplete states the algorithm generates a new set of membership queries, which are strings of symbols from the input-alphabet. This is done by determining an access sequence that leads from the hypothesis' initial state to the incomplete state under examination, and extending this sequence with every symbol of the input-alphabet, resulting in k queries. The access sequence can e.g. be determined using a standard Dijkstra search [3] on the current hypothesis or via any other search strategy.

Once a state is completed, there are two possibilities:

1. If it happens to have the same output signature as an already present complete state both stat es are considered equivalent and "merged" as illustrated in the step from 4 to 5 of Figure 2.
2. Otherwise, new successor states are created for each input symbol and enqueued as incomplete states in the worklist. This step clearly reveals the breadth-first exploration scheme, ensuring that any edge redirected to a previous state with identical signature is then pointing at the same or lower search level.

The sibling-based completion terminates once the queue of incomplete states is empty.

Figure 2 shows the first steps of the DHC algorithm. The hypothesis at first includes only the incomplete initial state, which is completed using membership queries. This results in new incomplete states, that are completed using additional queries. In this example both successors of the initial state show the same output behavior after completion, which causes them to be merged.

A pseudocode representation of the core algorithm is given in Figure 3. The method `getAccessSequence` returns a sequence of input symbols which reaches the respective state from the initial state of the hypothesis. The `doMembershipQuery` method expects a sequence of input symbols and returns the system response by means of a membership query. An eventual sibling to the current state will be retrieved by the `findStateWithSameSignature` method. If a sibling is found, the current state will be removed from the hypothesis by the `remove` method, after ensuring any transitions are redirected to the detected sibling by invoking `rerouteAllTransitions`. If no sibling was found, `createSuccesorsForEveryTransition` will create new states for every transition of the retained state, which subsequently are enqueued for exploration.

3.2 Step 2: Refining the Input Alphabet

The second step deals with the case when the equivalence oracle returns a counterexample. A simple way to treat a counterexample is to add all the suffixes of the counterexample to the input-alphabet and re-start the learning with the extended alphabet. This resembles the approach presented in [8] for guaranteeing that the newly started breadth-first exploration will take into account the knowledge about the diverging behavior inherent in the counterexample.

A more efficient approach to counterexample treatment analyzes the counterexample in order to determine a suffix d which separates two previously assumed to be equivalent states[16]. This can be achieved using a binary search on the counterexample and some additional MQs to pinpoint the diverging behavior [17]. The DHC algorithm then proceeds with step 1 after adding the suffix d as a new (artificial) symbol to the input alphabet. This guarantees that each additional input symbol leads to at least one more state. Table 1 illustrates the impact of this improved counterexample treatment according to a number of different methods explained in the next Section. As we see comparing the first with the last row, the gains are significant and vary from a factor 10 to a factor 30.

After extending the alphabet in one of these ways the breadth-first exploration could proceed as described in the previous section. The artificial input symbols, which we call *splitters*, are indeed sequences of input symbols only introduced to split states previously considered equivalent. Splitters are somewhat different from the normal alphabet symbols. They should, e.g., not be represented in the final result presented to a user, and they need not be considered when filling the worklist after having detected a new state as will be discussed in Section 4. Figure 4 illustrates the effect of splitters.Splitters are indicated by the square

```
1  function DHC(Alphabet alphabet) {
2    Hypothesis hypo = new Hypothesis();
3    Queue statesToComplete = new Queue();
4    statesToComplete.enqueue(hypo.getStartState());
5
6    while(statesToComplete.isNotEmpty()) {
7      State currentState = statesToComplete.dequeue();
8      Sequence accessSeq = currentState.getAccessSequence();
9
10     for(Symbol sym in alphabet) {
11       Query query = accessSeq.append(sym);
12
13       // Communicate with the target system, fetch the output symbol
14       Symbol output = doMembershipQuery(query);
15
16       // set the transition output for the sym input-symbol
17       // to the retrieved output symbol.
18       currentState.setTransitionOutput(sym, output);
19     }
20
21     State sibling = hypo.findStateWithSameSignature(currentState);
22     if(exists(sibling)) {
23       // reroute all transitions to currentState to sibling
24       hypo.rerouteAllTransitions(currentState, sibling);
25       hypo.remove(currentState);
26     } else {
27       currentState.createSuccesorsForEveryTransition();
28       for(State successor of currentState) {
29         statesToComplete.enqueue(successor);
30       }
31     }
32   }
33
34   return hypo;
35 }
```

Fig. 3. Pseudocode of the DHC core algorithm

brackets in the left graph (for instance, the splitter “[a,a]” joined by output such as “[0,1]”) and removed in the right graph, the actual learning result.

4 Notes on Efficient Implementation Strategies

While the DHC algorithm itself is simple and can be implemented with little effort, there are some hurdles to overcome to make the implementation actually perform well. The following suggestions employ standard algorithm-engineering approaches. To evaluate the impact of the proposed implementation strategies,

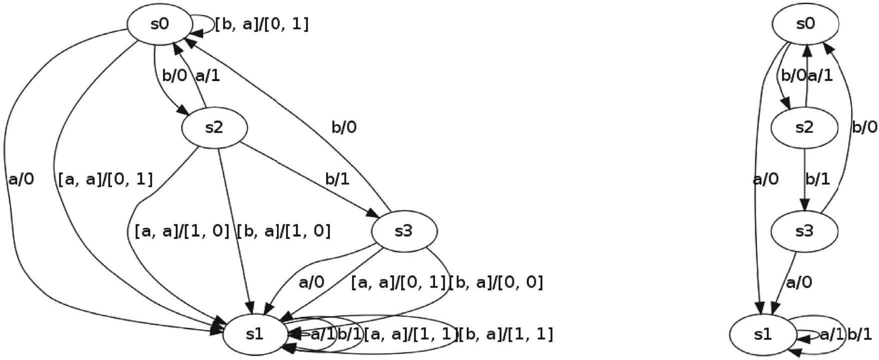


Fig. 4. The effect of splitters

Table 1. Comparison of the impact of the optimizations

Split	Ref	Sib	49 states	66 states	88 states
			17.90 s	49.59 s	353.41 s
		✓	10.70 s	30.67 s	210.04 s
	✓		8.95 s	23.21 s	162.42 s
	✓	✓	2.03 s	5.08 s	20.92 s
✓			13.68 s	29.86 s	181.69 s
✓		✓	8.42 s	18.78 s	111.10 s
✓	✓		6.77 s	13.96 s	79.89 s
✓	✓	✓	1.70 s	3.44 s	11.39 s

we conducted experiments with an implementation [10] that allows enabling and disabling specific optimizations.

Exploring only successor states of original alphabet symbols: As can be observed in Figure 4, states reached by splitter transitions are also reached by successively following the transitions of the splitters' individual symbols. This is guaranteed to be the case, as splitters are unrolled into sequences of individual symbols on membership query construction, which generates identical traces to successively following transitions of the individual symbols. Thus, as splitter transitions do not reach states not reached otherwise, it is not necessary to to enqueue any successor state of splitter transitions. For the provided example automaton, for instance, this decreases the number of generated membership queries from 78 to 46, the same number the algorithm of [7] would require when following the same path of construction.

Adding only one splitter per counterexample (Split): To complete a state, one membership query per alphabet symbol will be generated, meaning that the num-

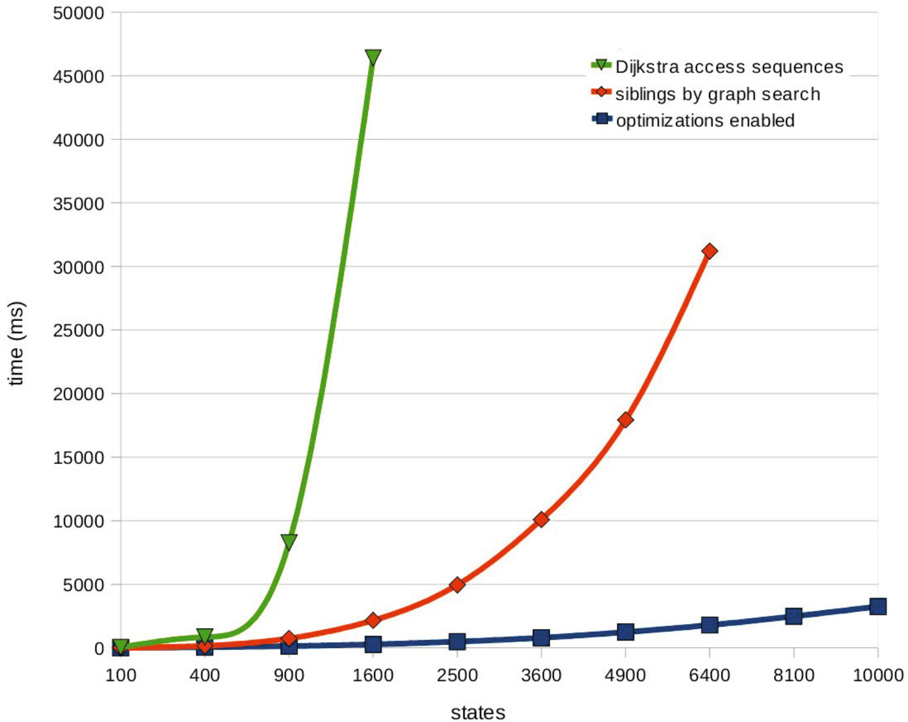


Fig. 5. The impact of the optimization strategies on scalability

ber of MQs generated by the DHC learning algorithm depends on the effective size of the alphabet, that includes the original input-alphabet Σ and the splitters added during the enhancement of the signatures. As described in Sect. 3.2, it is possible to analyze counterexamples so that only one splitter needs to be added to the alphabet, which drastically decreases the size of the required signatures, and consequently the number of membership queries.

The impact of this optimization is presented in Table 1, which shows that employing this optimization, denoted as *Split* in the table, decreases the overall runtime for all examined examples.

Determining access sequences quickly (Ref): For every incomplete state we need to determine an access sequence for the construction of membership queries. A straightforward approach would be to, e.g., employ a Dijkstra search, which is a safe bet and will work in all cases. Figure 5, however, shows that the Dijkstra search (not surprisingly) scales extremely poorly with increasing state count.

This can be overcome by maintaining the reverse edges of the spanning tree when constructing the DHC hypothesis, which immediately allows one to collect the access sequence in linear time. Table 1 illustrates the impact of this optimization here indicated as *Ref*.

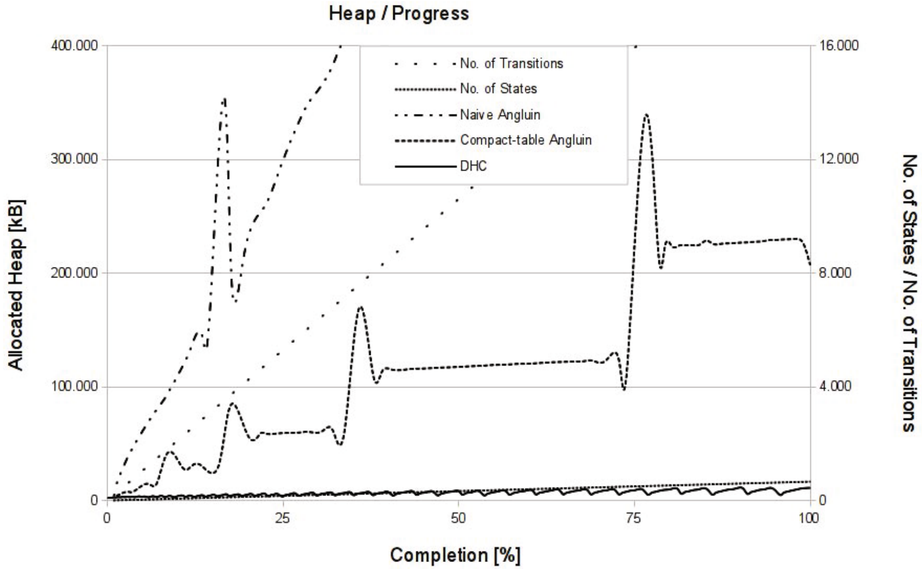


Fig. 6. Memory consumption of DHC compared to two algorithms with observation tables

Finding siblings fast (Sib): Finding siblings by searching the complete hypothesis becomes increasingly slower as the hypothesis grows as is shown in Figure 5, where employing a graph-wide search for siblings scales unfavorably with increasing state counts.

A good optimization is to sort all states with an unique output signature into a decision tree, which guarantees a lookup time solely depending on the input-alphabet of the hypothesis. The immense impact of this optimization is visible in Figure 5, where this optimization is included in *optimizations enabled*, and also in Table 1, where this optimization is referred to as *Sib*.

The overall impact of our optimizations is summarized in Figure 5, which in particular shows how nicely the optimized algorithm scales with growing state spaces.

5 Conclusion

In this paper we have presented an automata learning algorithm that follows the well-known breadth-first-oriented exploration pattern and is thus, due to its continuous provision of a visualizable hypothesis structure, fit for educational purposes. Moreover, these DHC hypotheses, the main data structures of the DHC algorithm, are significantly smaller than observation table: they grow with $O(nk)$, whereas observation tables grow with $O(nk^2 + n^2k)$ already in the case of

[7], and the estimation for L^* has even an additional factor "maximal size of a counterexample". This difference is nicely illustrated in Fig. 6, which clearly indicates that the memory consumption of table-based algorithms correlates with the number of discovered transitions (coarsely dotted line), whereas DHC's memory consumption only grows linearly with the number of discovered states (finely dotted line). [1]

Although this memory efficiency comes at the price of recomputation during each iteration, the DHC algorithm allowed us to raise the record for active learning to systems with more than a million states in settings where membership queries are extremely fast. The DHC algorithm is implemented in the LearnLib [10,9], which is available as free download at `http://www.learnlib.de`.

A closer investigation reveals that the DHC algorithm can be considered as an elegant means for splitting the data maintained in the observation table in two parts, the ones to be kept in main memory, namely the DHC hypotheses, and data that can be kept on disk or SSD, namely a cache, storing the results of all membership queries. This extends the practical impact of DHC's reduction of the memory footprint to large systems where answering membership queries is (very) expensive:

- It allows one to deal with systems of a few hundred thousand states whose observation table would grow far beyond the available main memory.
- The additional cache lookups required for the DHC algorithm do not weight in comparison to the membership querying times.

This separation of concerns is quite similar to the one proposed in [5,6,18] for externalizing the bulk of the required memory for graph search. We are convinced that following this line of thought that considers layered memory hierarchies can be quite successful also for active automata learning, even though their work depends on that fact that the considered graphs are provided in a white box fashion. This excludes the direct use of GPUs as proposed by [18], but it seems to have nevertheless a great potential for parallelization. First results in this direction are reported in [4].

References

1. Aarts, F., Jonsson, B., Uijen, J.: Generating Models of Infinite-State Communication Protocols Using Regular Inference with Abstraction. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 188–204. Springer, Heidelberg (2010)
2. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)

[1] The large spikes seen for the table-based algorithms are caused by copying data from fixed-size data structures into freshly allocated data structures of bigger size, and the ripples in DHC's memory profile can be explained by the from-scratch construction principle whenever a counterexample is delivered, which leads to the deallocation of memory consumed by the disproved hypothesis.

3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
4. Howar, F., Bauer, O., Merten, M., Steffen, B., Margaria, T.: The Teachers' Crowd: The Impact of Distributed Oracles on Active Automata Learning. In: Hähnle, R., et al. (eds.) ISoLA 2011 Workshops. CCIS, vol. 336, pp. 232–247. Springer, Heidelberg (2012)
5. Jabbar, S.: External directed search. KI 21(1), 37–38 (2007)
6. Jabbar, S.: External memory algorithms for state space exploration in model checking and action planning. PhD thesis (2008)
7. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
8. Maler, O., Pnueli, A.: On the Learnability of Infinitary Regular Sets. Information and Computation 118(2), 316–326 (1995)
9. Merten, M., Howar, F., Steffen, B., Cassel, S., Jonsson, B.: Demonstrating Learning of Register Automata. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 466–471. Springer, Heidelberg (2012)
10. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next Generation LearnLib. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011)
11. Nerode, A.: Linear Automaton Transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
12. Niese, O.: An Integrated Approach to Testing Complex Systems. PhD thesis, University of Dortmund, Germany (2003)
13. Raffelt, H., Margaria, T., Steffen, B., Merten, M.: Hybrid test of web applications with webtest. In: TAV-WEB 2008: Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, pp. 1–7. ACM, New York (2008)
14. Raffelt, H., Merten, M., Steffen, B., Margaria, T.: Dynamic testing via automata learning. Int. J. Softw. Tools Technol. Transf. 11(4), 307–324 (2009)
15. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. Int. J. Softw. Tools Technol. Transf. 11(5), 393–407 (2009)
16. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. 103(2), 299–347 (1993)
17. Steffen, B., Howar, F., Merten, M.: Introduction to Active Automata Learning from a Practical Perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)
18. Sulewski, D., Edelkamp, S., Kissmann, P.: Exploiting the computational power of the graphics card: Optimal state space planning on the gpu. In: Bacchus, F., Domshlak, C., Edelkamp, S., Helmert, M. (eds.) ICAPS. AAAI (2011)

Author Index

Aßmann, Uwe 1

Bauer, Oliver 232
Bennaceur, Amel 220
Breu, Ruth 162
Bruckner, Dietmar 144, 156
Brugali, Davide 46
Bruyninckx, Herman 46

Eilers, Sönke 61
Einramhof, Peter 106

Felderer, Michael 162

Gander, Matthias 162
Gelbard, Friedrich 144
Gherardi, Luca 46
Götz, Sebastian 1
Gurov, Dilian 181

Hinchey, Mike 91
Hinterleitner, Isabella 156
Howar, Falk 232, 248

Issarny, Valérie 220

Johansson, Richard 220

Katt, Basel 162
Klotzbücher, Markus 46
Kuka, Christian 61
Kulk, Jason 31

Leuthäuser, Max 1

Margaria, Tiziana 232, 248
McDonald, John 16
Meinke, Karl 200
Merten, Maik 232, 248
Middleton, Richard H. 16
Moschitti, Alessandro 220

Niu, F. 200

Østvold, Bjarte M. 181
Olufs, Sven 106

Pollhammer, Klaus 136
Pongratz, Martin 136
Ponweiser, Wolfgang 121

Reimann, Jan 1
Ruehrup, Stefan 61

Schaefer, Ina 181
Schreiner, Dietmar 150
Schroeter, Julia 1
Schwarz, Robert 106
Schweigert, Sören 61
Sindhu, M. 200
Spalazzese, Romina 220
Steffen, Bernhard 232, 248
Stüdli, Sonja 16
Sykes, Daniel 220
Szep, Alexander 136

Toben, Tobe 61

Varadarajan, Karthik 106
Vassev, Emil 91
Vincze, Markus 106, 121, 156

Welsh, James S. 31
Wende, Christian 1
Whelan, Thomas 16
Wilke, Claas 1
Winkelmann, Hannes 61
Wohlkinger, Walter 106
Wotawa, Franz 76

Zillich, Michael 121