ADAPTIVE 2013

The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications

ISBN: 978-1-61208-274-5

May 27- June 1, 2013

Valencia, Spain

ADAPTIVE 2013 Editors

Jaime Lloret Mauri, Polytechnic University of Valencia, Spain

Jose Alfredo F. Costa, Universidade Federal do Rio Grande do Norte (UFRN), Brazil

ADAPTIVE 2013

Foreword

The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2013), held between May 27 and June 1, 2013 in Valencia, Spain, targeted advanced system and application design paradigms driven by adaptiveness and self-adaptiveness. With the current tendencies in developing and deploying complex systems, and under the continuous changes of system and application requirements, adaptation is a key feature. Speed and scalability of changes require self-adaptation for special cases. How to build systems to be easily adaptive and self-adaptive, what constraints and what mechanisms must be used, and how to evaluate a stable state in such systems are challenging duties. Context-aware and user-aware are major situations where environment and user feedback is considered for further adaptation.

We take here the opportunity to warmly thank all the members of the ADAPTIVE 2013 Technical Program Committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to ADAPTIVE 2013. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the ADAPTIVE 2013 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that ADAPTIVE 2013 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of adaptive and self-adaptive systems and applications.

We are convinced that the participants found the event useful and communications very open. We hope that Valencia, Spain provided a pleasant environment during the conference and everyone saved some time to explore this historic city.

ADAPTIVE 2013 Chairs:

ADAPTIVE General Chair
Jaime Lloret Mauri, Polytechnic University of Valencia, Spain

ADAPTIVE Advisory Chairs
Radu Calinescu, Aston University, UK
Thomas H. Morris, Mississippi State University, USA
Serge Kernbach, University of Stuttgart, Germany
Antonio Bucchiarone, FBK-IRST of Trento, Italy
Jose Alfredo F. Costa, Universidade Federal do Rio Grande do Norte (UFRN), Brazil

ADAPTIVE Industry/Research Chairs
Dalimír Orfánus, ABB Corporate Research Center, Norway
Weirong Jiang, Juniper Networks Inc. - Sunnyvale, USA

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

ADAPTIVE 2013

Committee

ADAPTIVE General Chair

Jaime Lloret Mauri, Polytechnic University of Valencia, Spain

ADAPTIVE Advisory Chairs

Radu Calinescu, Aston University, UK
Thomas H. Morris, Mississippi State University, USA
Serge Kernbach, University of Stuttgart, Germany
Antonio Bucchiarone, FBK-IRST of Trento, Italy
Jose Alfredo F. Costa, Universidade Federal do Rio Grande do Norte (UFRN), Brazil

ADAPTIVE Industry/Research Chairs

Dalimír Orfánus, ABB Corporate Research Center, Norway
Weirong Jiang, Juniper Networks Inc. - Sunnyvale, USA

ADAPTIVE 2013 Technical Program Committee

Sherif Abdelwahed, Mississippi State University, USA
Nadia Abchiche-Mimouni, Université d'Evry, France
Habtamu Abie, Norwegian Computing Center/Norsk Regnesentral-Blindern, Norway
Muhammad Tanvir Afzal, Mohammad Ali Jinnah University- Islamabad, Pakistan
Jose Maria Alcaraz Calero, Hewlett-Packard Laboratories-Bristol, UK
Giner Alor Hernández, Instituto Tecnológico de Orizaba - Veracruz, México
Richard Anthony, University of Greenwich, UK
Flavien Balbo, Université Paris-Dauphine, Lamsade-CNRS, France
Luciano Baresi, Politecnico di Milano, Italy
Imen Ben Lahmar, Institut Telecom SudParis, France
Jesus G. Boticario, Centre for Innovation and Technological Development UNED, Spain
Sven Brueckner, Soar Technology, Inc., USA
Yuriy Brun, University of Washington - Seattle, USA
Radu Calinescu, Aston University, UK
Aldo Campi, Center for Industrial Research on ICT (CIRI ICT) - University of Bologna., Italy
Valérie Camps, IRIT-Toulouse, France
Radu Calinescu, University of York, UK
Chris Cannings, University of Sheffield, UK
Carlos Carrascosa, Universidad Politécnica de Valencia, Spain
Federica Cena, University of Torino, Italy
Luke Chen, University of Ulster, UK
Po-Hsun Cheng, National Kaohsiung Normal University, Taiwan

Paulo Martins, University of Trás-os-Montes e Alto Douro (UTAD), Portugal
Olga Melekhova, Université Pierre et Marie Curie - Paris 6, France
John-Jules Meyer, Universiteit Utrecht, The Netherlands
Frederic Migeon, IRIT/Toulouse University, France
Gero Müehl, University of Rostock, Germany
Christian Müller-Schloer, Leibniz University of Hanover, Germany
Masayuki Murata, Osaka University Suita, Japan
Filippo Neri, University of Naples "Federico II", Italy
Dirk Niebuhr, Clausthal University of Technology, Germany
Andrea Omicini, Università degli Studi di Bologna - Cesena, Italy
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Mathias Pacher, Leibniz Universität Hannover, Germany
Raja Humza Qadir, dSPACE GmbH, Paderborn, Germany
Claudia Raibulet, University of Milano-Bicocca, Italy
Mahesh (Michael) S. Raisinghani, TWU School of Management, USA
Sitalakshmi Ramakrishnan, Monash University, Australia
Wolfgang Reif, University of Augsburg, Germany
Brian M. Sadler, Army Research Laboratory, USA
Yacine Sam, Université François Rabelais Tours, France
Huseyin Seker, De Montfort University Leicester, UK
Sebastian Senge, TU Dortmund, Germany
Igor Sfiligoi, University of California San Diego - La Jolla, USA
Vasco Soares, Instituto de Telecomunicações / Polytechnic Institute of Castelo Branco, Portugal
Christoph Sondermann-Wölke, Universität Paderborn, Germany
Panagiotis Spapis, National and Kapodistrian University of Athens, Greece
Stephan Stilkerich, EADS Innovation Works, Germany
Greg Sullivan, BAE Systems, USA
Yehia Taher, Tilburg University, The Netherlands
Javid Teheri, The University of Sydney, Australia
Christof Teuscher, Portland State University, USA
Sotirios Terzis, University of Strathclyde, UK
Catherine Tessier, ONERA - Toulouse, France
Christof Teuscher, Portland State University, USA
Peppo Valetto, Drexel University, USA
Arlette van Wissen, VU University Amsterdam, Netherlands
Natalie van der Wal, VU University Amsterdam, Netherlands
Eiko Yoneki, University of Cambridge, UK

Table of Contents

On the Utilization of Heterogeneous Sensors and System Adaptability for Opportunistic Activity and Context Recognition

Marc Kurz, Gerold Hölzl, Alois Ferscha
Johannes Kepler University Linz
Institute for Pervasive Computing
Linz, Austria
{kurz, hoelzl, ferscha}@pervasive.jku.at

***Abstract*—Opportunistic activity and context recognition systems draw from the characteristic to utilize sensors as they happen to be available instead of predefining a fixed sensing infrastructure at design time of the system. Thus, the kinds and modalities of sensors are not predefined. Sensors of different types and working characteristics shall be used equally if the delivered environmental quantity is useful for executing a recognition task. This heterogeneity in the sensing infrastructure and the lack of a defined sensor infrastructure motivates the utilization of sensor abstractions and sensor self-descriptions for identifying and configuring sensors according to recognition tasks. This paper describes how sensors of different kinds can be accessed in a common way, and how they can be utilized at runtime by using their semantic self-descriptions. The different steps within the lifecycle of sensor descriptions are described to understand the powerful concepts of self-describing sensors and sensor abstractions. Furthermore, a prototypical framework realizing the vision of opportunistic activity recognition is presented together with a discussion of subsequent steps to adapt the system to different application domains.**

***Keywords*—Activity recognition; system adaption; opportunistic activity recognition; heterogeneous sensors**

I. INTRODUCTION

Common and established activity and context recognition systems usually define the recognition task together with the sensing infrastructure (i.e., the sensors, their positions and locations, spatial and proximity relationships, sampling rates, etc.) initially, at design time of the system. The successful recognition of activities and more generally the context of subjects is heavily dependent on the reliability of the sensing infrastructure over a certain amount of time, which is often difficult to achieve, due to sensor displacements or sensor disconnects (e.g., a sensor may run out of power). In contrast to that, opportunistic systems utilize sensor systems as they happen to be available to execute a dynamically defined recognition goal [1][2]. The challenge altered from deploying application specific sensor systems for a fixed recognition task to the utilization of sensors that happen to be available for dynamically stated recognition goals [1][3][4]. The available sensor systems have to be discovered, identified, and configured to cooperative sensor *ensembles* that are best suited to execute a certain recognition goal in a specific application domain. Furthermore, an opportunistic system has to be robust and flexible against spontaneous changes in the surrounding sensor environment, allowing the continuity of the recognition process even if sensors disappear (or appear) in the sensing infrastructure [5]. Therefore, three crucial challenges (amongst others) can be identified: (i) the utilization of sensor systems of different kinds and modalities as data delivering entities, (ii) the identification of sensors and their capabilities for configuring ensembles according to recognition goals, and (iii) the adaptation of an opportunistic activity recognition system (together with the sensor representations and the low-level algorithmic dependencies) to a specific application domain. This paper presents the concepts of sensor abstractions [1][6] and sensor self-descriptions [1] to cope with these challenging aspects. Furthermore, a reference implementation of an opportunistic activity and context recognition system is presented, referred to as the *OPPORTUNITY Framework* [1][6][7] accompanied with a discussion how the framework together with the sensor representations (composed of abstractions and self-descriptions) can be easily adapted to diverse application domains.

The remainder of the paper is structured as follows. Section II motivates and presents the concept of sensor abstractions, which enables a common usage of different sensor systems. Section III describes how sensor systems can be utilized and configured dynamically according to an actual recognition goal by using their self-description, and how the sensor self-description evolves over time by illustrating the self-description life-cycle. Section IV discusses how the OPPORTUNITY Framework can be adapted to different application domains. The final Section V closes with a conclusion and summarizes the core contributions of this paper.

II. SENSORS IN THE OPPORTUNITY FRAMEWORK

Opportunistic activity and context recognition systems do not predefine their sensing infrastructure initially, as it was the usual case in decades of related systems (e.g., *Salber et al.* [8], *Bao and Intille* [9], *Ravi et al.* [10], *Tapia et al.* [11], and *Ward et al.* [12]). Instead, the system makes best use of the currently available sensors for executing a recognition goal. This aspect also includes heterogeneity within the sensing infrastructure, as the lack of a defined sensing infrastructure also includes missing definitions of the kinds and modalities of the sensors involved in an ensemble. Therefore, an activity recognition system that operates in an opportunistic way has to be capable of handling different sources of environmental data.

 ISBN: 978-1-61208-274-5

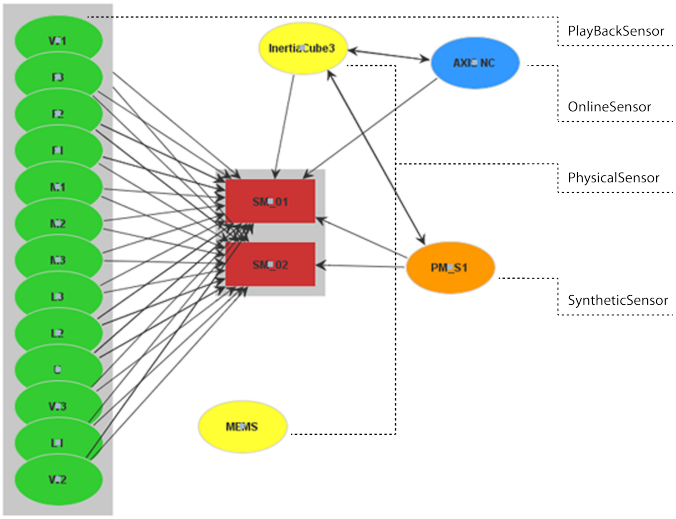


Fig. 1. An actual sensing infrastructure showing different types of available sensors.

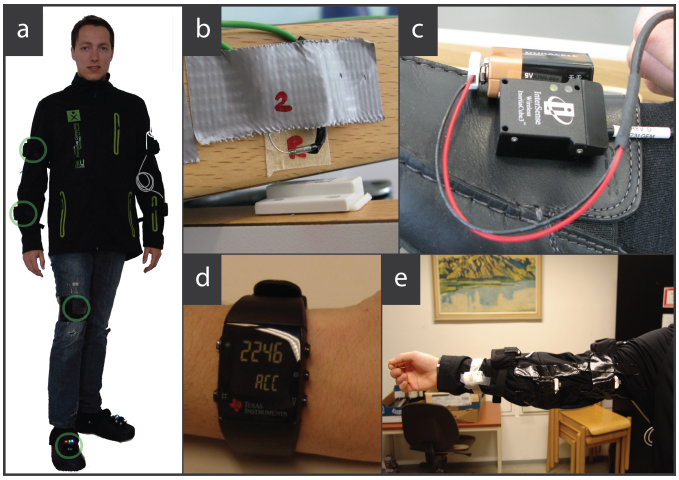


Fig. 2. Impressions of the (physical) sensor systems that are available within the OPPORTUNITY Framework.

These sources do not have to necessarily be physical sensors (e.g., acceleration, orientation, temperature, etc.), but can also be immaterial devices that can provide valuable information to a system [6]. Sensor abstractions [1][6] provide a common and easy accessible interface to handle different kinds of material and immaterial devices as general type *Sensor* (e.g., physical, online, playback, synthetic, and harvest sensors). The abstractions hide the low level access and connection details and provide methods to handle different devices in a common way. This concept enables the inclusion of sensors in ensemble configurations (the set of sensors that is best suited to execute a recognition goal [2][4]) of different kinds, types and modalities as they happen to be available.

The OPPORTUNITY Framework [1][7] is a prototypical implementation (written in Java/OSGi) of a system that recognizes human activities in an opportunistic way. By enabling the utilization of sensors of different modalities, thus does not restrict the sensing infrastructure to be composed of a predefined set of specific sensors, the system is flexible towards the generation of ensembles for activity recognition. By further utilizing the concept of self-describing sensors (see Section III) the system is robust against changes in the sensing infrastructure, thus can react on spontaneous changes on the sensors' availability by reconfiguring the corresponding activity recognition chains and the ensemble [5]. Furthermore, since also immaterial devices like a *PlayBackSensor* [6] - that replays a pre-recorded data source, thus simulates an actual sensor - can be utilized at runtime of the system, this allows the configuration of hybrid simulation scenarios made of physical and simulated (playback) devices. The different classes that implement the hardware access (in case of *PhysicalSensors*), the connection to a remote data source (*OnlineSensor*), or the reading of datasource for *PlayBackSensors* are all derived from a common interface. This means from the framework's point of view all these devices and sources of environmental data can be accessed and utilized in a common way.

Figure 1 displays an example for an actual sensing infrastructure with two active recognition goals (the two red rectangles) within the OPPORTUNITY Framework. This schematic illustration is available as visualization in the OPPORTUNITY Framework and presents the current available sensor devices, the active sensing mission, and the active data flows between the involved units. The entire actual sensing infrastructure in this example consists of 17 sensors, each illustrated by a colored ellipse, whereas 13 are of type *PlaybackSensor* (green), 2 are of type *PhysicalSensor* (yellow), respectively one of type *OnlineSensor* (blue), and one of type *SyntheticSensor* (orange). The arrows in the figure indicate the dataflows from sensors to active recognition goals, and between sensors themselves. Thus, an ensemble is the best configurable set of sensors that cooperates to execute a recognition goal, whereas different types of sensors can be utilized by accessing them in a common, standardized way by providing interfaces and APIs to hide the low-level access details. The following Table I provides an overview of the currently available sensor abstractions in the OPPORTUNITY Framework.

The data sources for the sensors of type *PlaybackSensor* in Table I have been recorded in two recording sessions. First, a kitchen scenario in May 2010 was set up, where 72 sensors with more than 10 modalities have been utilized, and 12 subjects performed early morning activities, each 6 runs. Second, another kitchen equipped with sensors in December 2011, where 5 subjects performed activities like *coffee preparation*, *coffee drinking*, and *table cleaning*. These recording sessions are described in detail in [13] and [3]. Figure 2 provides impressions of the sensors that are made available as *PlaybackSensor* or *PhysicalSensors* in the OPPORTUNITY Framework. This means, they can be replayed anytime and behave as if they would be physically present. This enables the configuration of hybrid and powerful simulation scenarios for opportunistic activity recognition.

Figure 2(a) shows the *MotionJacket* sensor [14], which contains five XSens MTx units, mounted on the upper and lower arms, and one on the upper back position. Furthermore, one bluetooth accelerometer is mounted on the knee of the person, and one SunSPOT device is attached on the shoe toe-box. Figure 2(b) displays a reed switch as it was used in the dataset recording session in [13]. These magnetic switches were mounted in the environment, on different fitments and

 ISBN: 978-1-61208-274-5

TABLE I. OVERVIEW OF CURRENTLY AVAILABLE SENSOR ABSTRACTIONS IN THE OPPORTUNITY FRAMEWORK.

Short Name	*Sensor Type*	*# of Sensors*	*Further Details*
Reed Switch	PlaybackSensor	13	HAMLIN MITI-3V1 Magnetic Reed Switch
USB Accel	PlaybackSensor	8	USB ADXL330 3-axis Accelerometer
BT Accel	PlaybackSensor	1	Bluetooth ADXL330 3-axis Accelerometer
Ubisense	PlaybackSensor	1	UBISENSE Location Tracking System
Shoetoebox	PlaybackSensor	2	Sun SPOT LIS3L02AQ Accelerometer
Motionjacket	PlaybackSensor	5	XSENS Xbus Kit MTx
Motionjacket	PhysicalSensor	n	XSENS Xbus Kit MTx
Ubisense	PhysicalSensor	1 System (n tags)	UBISENSE Location Tracking System
TI Chronos	PhysicalSensor	n	Texas Instruments eZ430 Chronos
SunSpot	PhysicalSensor	n	Sun SPOT LIS3L02AQ Accelerometer
RFID	PhysicalSensor	n	Inside Contactless M210-2G
MEMS Microphone	PhysicalSensor	n	—
IPhone4	PhysicalSensor	n	Iphone4 Sensor Platform
IntertiaCube3	PhysicalSensor	n	InterSense Wireless InertiaCube3
TI EZ430	PhysicalSensor	n	Texas Instruments EZ430 Chronos
AxisCamera	OnlineSensor	n	AXIS 2120 Network Camera
FSA Pressure	SyntheticSensor	n	XSENSOR PX100:26.64.01

household appliances (e.g., drawers, fridges, doors, etc.). Figure 2(c) shows an InterSense Wireless Inertiacube3 capable of 3 DOF tracking (acceleration, gyro, and magnetometer), mounted on the shoes of persons. Clipping (d) of Figure 2 contains an off-the-shelf wrist worn device (i.e., the Texas Instruments EZ430 Chronos) in a watch-like form, that provides acceleration at a maximum sampling rate of 100Hz. The last clipping (e) shows multiple sensors as used in [13] and [3], and as made available in the OPPORTUNITY Framework as sensor abstraction. First, two of the XSens MTx systems (i.e., the *MotionJacket*) mounted on the upper and lower right arm are visible. Second, three of the bluetooth acceleration (the white devices) are shown. These self-constructed devices contain a simple acceleration sensor, a bluetooth communication unit and power supply.

The OPPORTUNITY Framework is meant to be open-ended. This means on the one hand that the abstraction concept is not restricted to the yet identified six abstractions (i.e., *PhysicalSensor*, *PlaybackSensor*, *OnlineSensor*, *SyntheticSensor*, *HarvestSensor*, and *ProxySensor*) [6]. Furthermore, the available sensors and sensor abstractions as presented in Table I and Figure 2 are a starting point in the OPPORTUNITY Framework and subject to add further (abstracted) sensors on demand. The following Section III describes the second important concept in opportunistic systems on the sensor level: *sensor self-descriptions*.

III. UTILIZING SENSORS

One major research challenge in an opportunistic activity recognition system is the fact that the sensor devices are not known at design time of the system. This means the system has to be able to handle devices of different modalities and kinds, and has to react on spontaneous changes in the sensing infrastructure. For enabling an opportunistic system to handle and access a possible variety of different devices and modalities - even material and immaterial devices - we discussed the concept of *Sensor Abstractions* in the previous Section II. Since not only the sensor infrastructure is subject to changes over time, but also the recognition goal is not defined in an opportunistic system, thus can be stated by users or applications at runtime [1][2][4], the set of sensors has to be identified that can be utilized for a recognition goal. This means that each sensor needs a description on a semantic level that provides the information to the system what it can be used for, how it has to be configured (e.g., which sensor signal features and classification algorithms have to be used, which parameters are required, etc.), and what is the expected performance. Therefore we propose the concept of *Sensor Self-Descriptions* providing information what the sensor can be used for and how it has to be configured [1].

The sensor self-description - as the name already tells - describes a sensor, thus provides relevant information about the physical and working characteristics and the recognition capabilities to the opportunistic activity and context recognition system. The description itself is tightly coupled to a sensor and has to meet different requirements, like (i) *machine-readability*, (ii) *human-readability*, (iii) *ease of access*, and (iv) *extensibility*. Taken these requirements, the decision about the format for the sensor self-descriptions is obvious: XML, respectively SensorML [15]. This XML language specification provides standard models, schemes and definitions for describing sensors and measurement processes.

The self-description of sensors is designed to semantically describe the sensing device on a meta-level regarding its working and physical characteristics (e.g., dimensions, weight, power consumption, sampling rate, etc.), and its recognition capabilities and assignment in sensor ensembles for specific recognition goals. These two use cases of the sensor self-descriptions emerge the need to segment them into one part of the description that holds the technical details as they are defined in the corresponding fact sheet delivered by the manufacturer, and into a second part that enables the dynamic configuration of the sensor in cooperative ensembles that aim at executing a recognition goal as accurate as possible. The dynamic part of the sensor self-descriptions contains so-called *ExperienceItems* (Figure 3 shows the important parts from an exemplary ExperienceItem, like the required classifier, the modality of the sensor, the location of the sensor, the recognizable activities together with the DoF value, and the required feature extraction method) [1][16].

Each ExperienceItem acts as snapshot to memorize the sensor capabilities in form of recognizable activities and further information about the sensor (e.g., location, orientation, topology of the place, etc.), thus describes a complete recognition chain [17] (i.e., *data preprocessing and segmentation*, *feature extraction*, *classification*, and *decision fusion*) and the specific methods. Each ExperienceItem features a correspond-

 ISBN: 978-1-61208-274-5

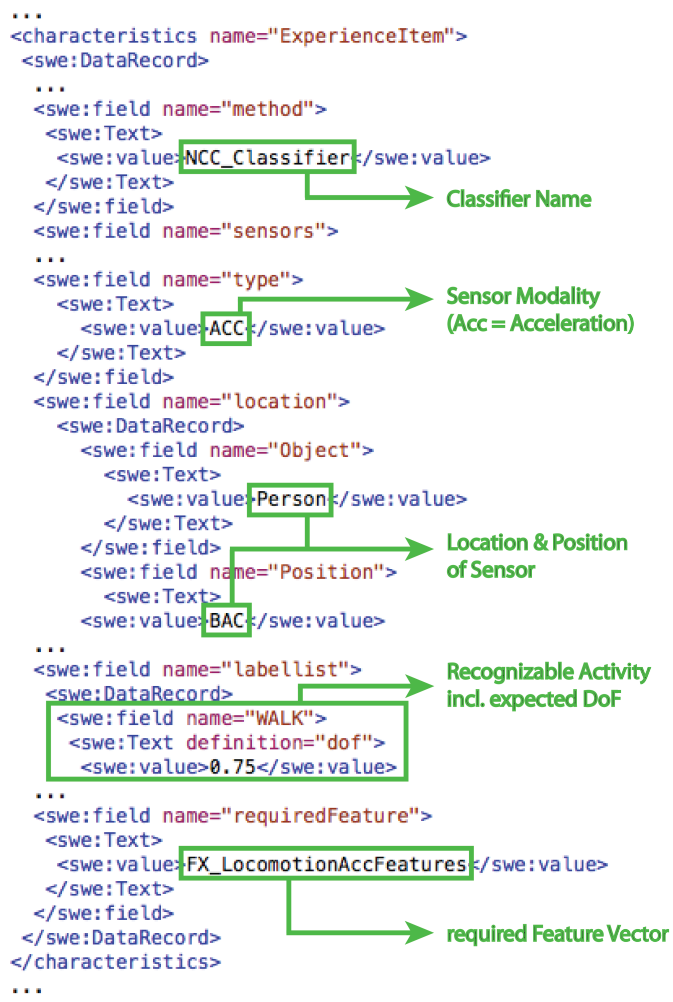


Fig. 3. Selected parts of an exemplary ExperienceItem as part of the sensor self-description [16].

ing *Degree of Fulfillment (DoF)*, which is a quality-of-service metric in the range of [0, 1], which expresses how well a certain activity is recognized (i.e., the *DoF* is an estimate of the expected accuracy) [1]. The ExperienceItem is used by the framework to configure an available sensor with the required machine learning algorithms and the correct training data (i.e., the complete activity recognition chain) to recognize a certain set of activities. ExperienceItems can either be generated offline by a human expert, or autonomously by the system at runtime. The manual generation of ExperienceItems requires offline labeling and training to gather a classifier model and the translation of the configured algorithms into SensorML, respectively self-description syntax. The more interesting way of generating ExperienceItems is done autonomously by the system by applying transfer learning (a sensor "learns" how to recognize certain activities from other sensors, experience is transferred to enhance the system's overall recognition capabilities) [14].

The segmented sensor self-description has different stages that can be described by the corresponding sensor lifecycle. Figure 4 shows the lifecycle for an exemplary sensor together with the stages and their transitions (i.e., (i) *sensor manufactured*, (ii) *sensor enrolled*, (iii) *expert knowledge*, (iv) *sensor active*, (v) *sensor ready*, and (vi) *sensor out of service*). The lifecycle-stages of the sensor and its self-description are described in the following list:

(i) *Sensor manufactured*: the sensor is ready to use and delivered with its technical specification. In Figure 4, the example on the left hand side shows an InterSense InertiaCube3 sensor with the corresponding datasheet. Neither the technical self-description, nor the dynamic description (in SensorML [15] syntax, as required in an opportunistic activity and context recognition system) is yet available at this stage in the lifecycle. The base for specifying and generating the technical self-description is given with the datasheet delivered with the device by the manufacturer.

(ii) *Sensor enrolled*: this stage in the sensor lifecycle occurs once the technical sensor self-description is available. This means the sensor is ready to be used within an opportunistic activity recognition system but still has no ExperienceItems in its dynamic description that enable the involvement in the execution of recognition goals.

(iii) *Expert knowledge*: this stage can be seen as extension to the previous stage (sensor enrolled). A human expert, who manually adds ExperienceItems to the dynamic sensor self-description, can extend the available (dynamic) self-description. This involves offline training and the manual extension of the dynamic sensor self-description by adding ExperienceItems.

(iv) *Sensor active*: The sensor is active, which means it is involved in the process of executing a recognition goal. The role of the sensor can either be that it is integrated in a running ensemble, or that it is involved as learner. That means its sensor self-description is extended autonomously by the system, by adding further ExperienceItems by observing the configured ensemble and its recognition results.

(v) *Sensor ready*: The sensor is ready to be used within the execution of specific recognition goals but is not currently involved in a running ensemble. That means its self-description already contains one or more ExperienceItems. In this passive mode, the enhancement of the self-description can be done again by a human expert in an offline way.

(vi) *Sensor out of service*: The sensor is outdated, which can be the case once a newer version of a specific sensor type is available. The corresponding self-description is versioned and made available for future use with the newer sensor device. The technical description might be outdated but the gathered experience in the dynamic sensor self-description could be of high value for the new device in future recognition goals.

The combination of the two concepts on the sensor level (i.e., sensor abstractions and sensor self-descriptions) represents a data delivering entity in an opportunistic activity recognition system. The step towards a whole new paradigm in

 ISBN: 978-1-61208-274-5

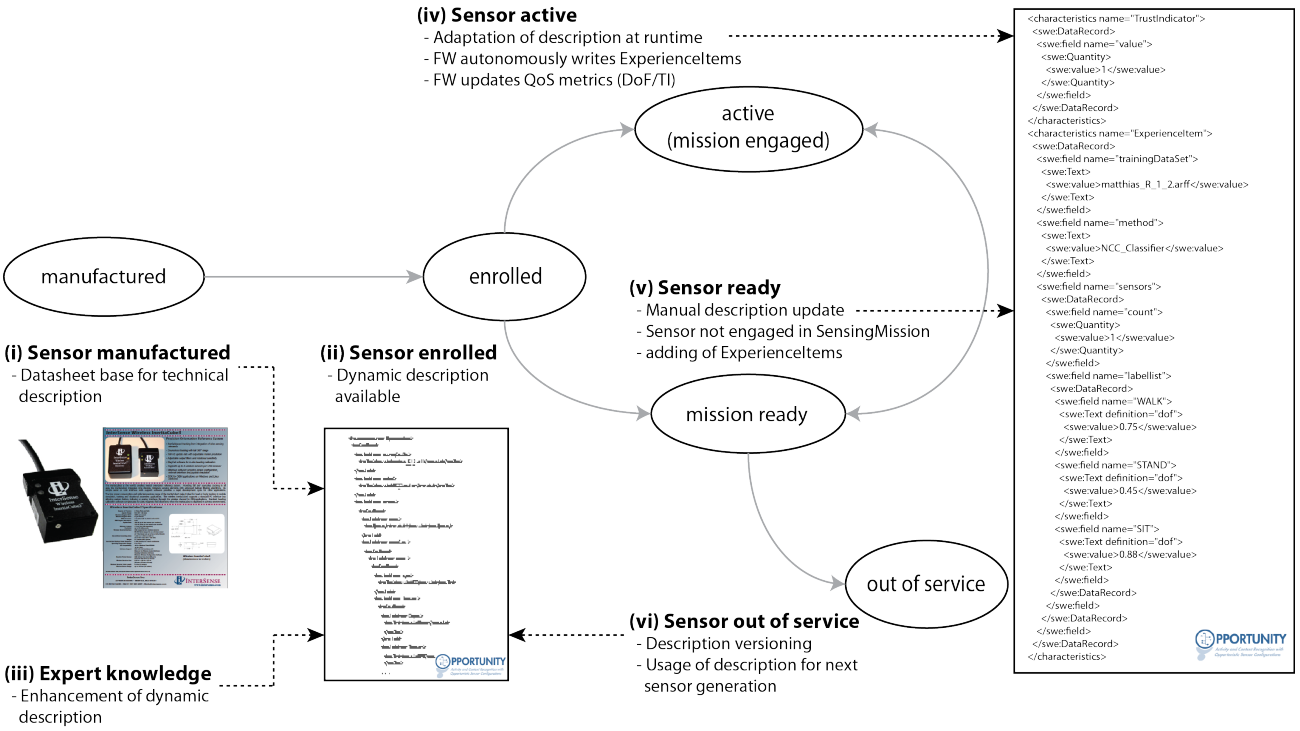


Fig. 4. The lifecycle of sensors and the corresponding self-descriptions in the OPPORTUNITY Framework.

activity recognition without the application dependent deployment of sensors and the reliability of the sensing infrastructure is done and opens new challenges (and possibilities) in open-ended systems. The foundation for flexible and robust activity recognition in an opportunistic way is given, the following Section IV discusses necessary steps to adapt the OPPORTUNITY Framework (and the accompanying sensor representations) to different application domains.

IV. Framework Adaptation

The OPPORTUNITY Framework together with the machine learning technologies, the sensor representations and high-level goal processing concepts has to be adaptable to different domains with less possible efforts. The developed concepts operate application and domain independent, they have to be taken and adapted accordingly. Based on the available reference implementation or an already existing domain- or application specific adapted release of the framework, the adaption process itself consists of at most three independent steps:

(i) Activity knowledge extension or replacement.

(ii) Sensor system inclusion to enhance the set of possible and accessible sensors.

(iii) Extension of the sensors self-descriptions.

An example for a necessary framework adaptation is the deployment of the system in private households for activity recognition in order to implicitly control electronic devices. This adaptation of the OPPORTUNITY Framework for optimized energy consumption in private households is described in detail in [18]. There, a field study has been conducted to evaluate the energy saving potentials based on the inhabitants activities (e.g., if someone is not watching television, the TV set can be safely switched off). The OPPORTUNITY Framework (which was used for activity recognition) has been adapted accordingly to meet the requirements and characteristics in such an application. Sensor abstractions have been added to make the expected sensors (i.e., smart phones, wrist-worn accelerometers, Ubisense positioning sensors) available in the application. New sensor descriptions have been added and existing descriptions have been modified to represent the recognition capabilities based on activity representations and relations (in form of an OWL-ontology). The system was able to (i) run stable over a two-week period in each household and to (ii) handle dynamically varying sensor settings.

In the following, these three steps are described in detail, and Figure 5 presents an illustration of the adaptation-workflow. Either all of the steps are executed or a subset of them, which is shown in the figure, to come from the starting basis of the framework (left-hand side) to the domain-adapted framework (right-hand side), depending on the situation.

A. Knowledge Representation Modification

The activity knowledge representation is composed using the W3C standard language OWL. Its purpose is to describe activities, the relations among them and more generally the context for a specific application domain. It is left to the application developer how this knowledge is designed, whether it is for example following the development criteria of a taxonomy (strictly hierarchical), or other semantic structures (e.g., ontologies, topic maps, etc.). In [1] we present an example for an ontology, which provides activities, movable and environmental objects, as network of relations for a kitchen

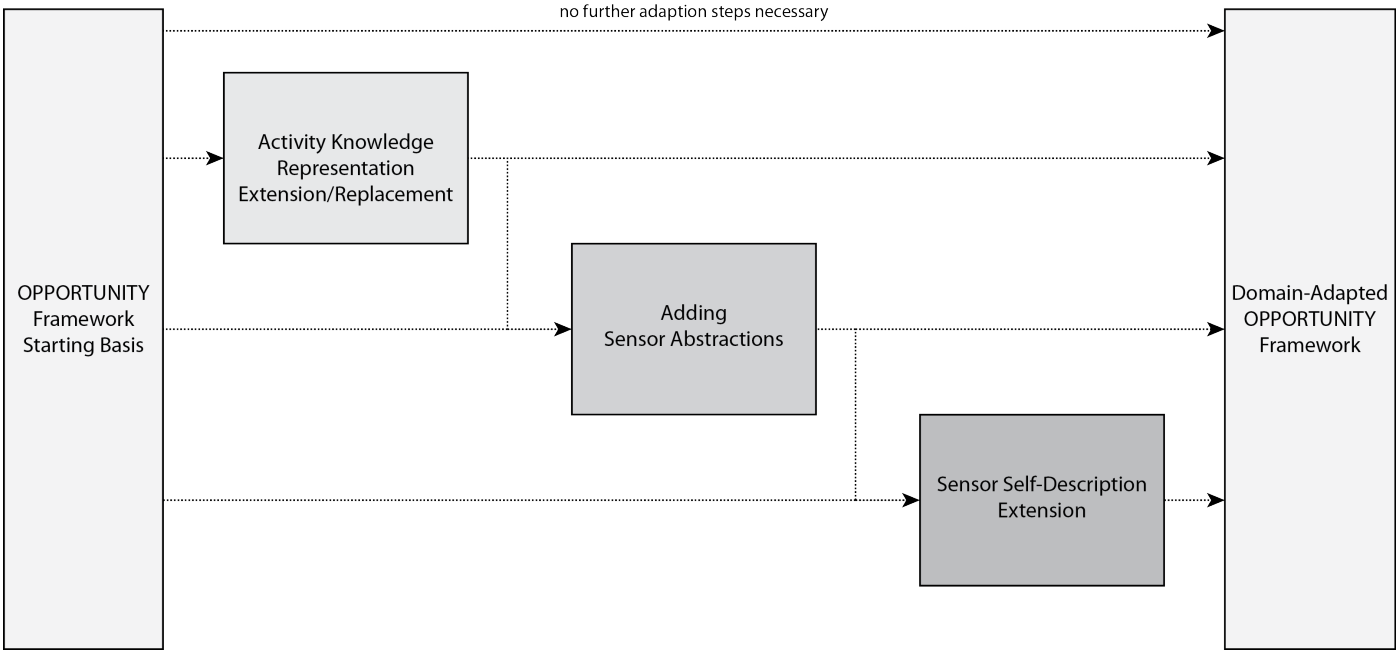


Fig. 5. The application-/domain-specific framework adaptation as three step process.

scenario, containing more than 130 different classes. The ontology itself builds the knowledge base for an application by providing a vocabulary and relations between terms explaining their relationship to each other.

B. Sensor System Inclusion

As already discussed, an opportunistic system does not restrict the kinds and modalities of sensors that act as input sources for environmental quantities. Therefore, to adapt the framework to a new domain, it might be necessary to add sensor abstractions to meet the requirements of possibly occurring sensors. This means, that an application developer, who adapts the framework has to add sensor abstractions by using the defined and common API in form of an interface that acts as common base for having a general way of accessing sensors. Once the abstraction for a sensor device is included in the framework, all appearing sensors of this type can be accessed equally and operate as general type *sensor*. The challenging aspects within the sensor system inclusion step are the low level access details, which have to be implemented once. From the framework's point of view - as all devices are derived from the interface that defines a sensor - those low level details of accessing the device are hidden. Not only material devices (e.g., acceleration, temperature, humidity, orientation sensors) are possible as sources of environmental quantities, but also immaterial sources, like online accessible webservices (e.g., weather or traffic information) can be of high value in an activity and context recognition system.

C. Self-Description Extension

The final step in the framework adaptation work flow is the extension of the sensor self-descriptions. If a completely new sensor type has been added in the previous step as new sensor abstraction, the inclusion of an accompanying new technical description is necessary (see Figure 4). This has to be done only once for each sensor type, since the technical description is static and shared among sensor of the same type. The modification of the dynamic sensor self-description can either make an extension of the existing descriptions and ExperienceItems necessary, or a definition of completely new dynamic descriptions. The first case occurs, whenever existing sensor devices are re-utilized for a new application domain. This makes the extension of the existing dynamic self-descriptions necessary to cover the new activity definitions according to the accompanying ontology by adding new ExperienceItems. The second case occurs, whenever new sensor devices are added and utilized in a new application (domain). This means, new dynamic self-descriptions have to be generated for each device initially. The extension of recognition capabilities in form of ExperienceItems can either be done before operation manually, or during runtime of the system autonomously (as described in [14]).

V. Conclusion and Future Work

This paper presents the two concepts of sensor abstractions and sensor self-descriptions that are big steps towards the vision of recognizing human activities in an opportunistic way (shown in a reference implementation called *OPPORTUNITY Framework*). The capability of utilizing heterogeneous devices by abstracting them to a generalized type - which can be of material and immaterial nature - enables flexible, continuous and dynamic activity recognition with presumably unknown sensor settings. The sensor self-descriptions provide semantic information about individual devices with respect to their capability of recognizing specific activities. This allows for (i) dynamically configuring activity recognition chains at system runtime, and (ii) to react on spontaneous changes in the sensing infrastructure in terms of appearing and disappearing sensor devices. The sensor (self-description) lifecycle and the stepwise adaptation of the OPPORTUNITY Framework to

 ISBN: 978-1-61208-274-5

specific application domains is discussed, whereas this can be broken down to three subsequent steps (i.e., (i) knowledge representation extension, (ii) sensor system inclusion, and (iii) self-description extension). The major contributions of this paper can be summarized to (i) the discussion and the proof of concept of the sensor representation composed of abstractions and self-descriptions, (ii) the identification of a sensor lifecycle representing the sensor's evolution over time, and (iii) - based on the previous items - the stepwise adaptation of an opportunistic activity recognition system to specific application domains.

Future work within the topic of utilizing heterogeneous sensors for accurate activity recognition will tackle the multi-sensor combination with sensor fusion technologies [19] for the specific activity classes. As discussed in related work (e.g., *Kuncheva* and *Whitacker* [20]), the prediction of the accuracy of multi-sensor combinations (i.e., ensembles) is a very challenging task. Currently, research work is conducted that utilizes the mutual information of pairwise sensor combinations in order to predict the accuracy of dynamically configured ensembles. Furthermore, shaping and optimization is currently investigated, meaning that the set of sensors that is included in an ensemble has to be well selected. If a desired activity can be recognized by a lot of sensors, including all of them in the ensemble does not necessarily mean that the accuracy is higher than including only a subset of sensors (the accuracy can even be worse). Therefore, the ensembles have to be optimized towards a maximized expected accuracy for the activities that have to be recognized.

ACKNOWLEDGMENT

The project OPPORTUNITY acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 225938.

REFERENCES

[1] M. Kurz, G. Hölzl, A. Ferscha, A. Calatroni, D. Roggen, G. Tröster, H. Sagha, R. Chavarriaga, J. del R. Millán, D. Bannach, K. Kunze, and P. Lukowicz, "The opportunity framework and data processing ecosystem for opportunistic activity and context recognition," *International Journal of Sensors, Wireless Communications and Control, Special Issue on Autonomic and Opportunistic Communications*, vol. 1, December 2011.

[2] D. Roggen, K. Förster, A. Calatroni, T. Holleczek, Y. Fang, G. Troester, P. Lukowicz, G. Pirkl, D. Bannach, K. Kunze, A. Ferscha, C. Holzmann, A. Riener, R. Chavarriaga, and J. del R. Millán, "Opportunity: Towards opportunistic activity and context recognition systems," in *Proceedings of the 3rd IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC 2009)*. Kos, Greece: IEEE CS Press, June 2009.

[3] G. Hölzl, M. Kurz, and A. Ferscha, "Goal oriented opportunistic recognition of high-level composed activities using dynamically configured hidden markov models," in *The 3rd International Conference on Ambient Systems, Networks and Technologies (ANT2012)*, August 2012.

[4] ——, "Goal processing and semantic matchmaking in opportunistic activity and context recognition systems," in *The 9th International Conference on Autonomic and Autonomous Systems (ICAS2013), March 24 - 29, Lisbon, Portugal*, March 2013, p. 7.

[5] M. Kurz, G. Hölzl, and A. Ferscha, "Dynamic adaptation of opportunistic sensor configurations for continuous and accurate activity recognition," in *Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE2012), July 22-27, Nice, France*, July 2012.

[6] M. Kurz and A. Ferscha, "Sensor abstractions for opportunistic activity and context recognition systems," in *5th European Conference on Smart Sensing and Context (EuroSSC 2010), November 14-16, Passau Germany*. Berlin-Heidelberg: Springer LNCS, November 2010, pp. 135–149.

[7] M. Kurz, A. Ferscha, A. Calatroni, D. Roggen, and G. Tröster, "Towards a framework for opportunistic activity and context recognition," in *12th ACM International Conference on Ubiquitous Computing (Ubicomp 2010), Workshop on Context awareness and information processing in opportunistic ubiquitous systems, Copenhagen, Denmark, September 26 - 29, 2010*, September 2010.

[8] D. Salber, A. Dey, and G. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*. ACM, 1999, pp. 434–441.

[9] L. Bao and S. Intille, "Activity recognition from user-annotated acceleration data," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, A. Ferscha and F. Mattern, Eds. Springer Berlin / Heidelberg, 2004.

[10] N. Ravi, D. Nikhil, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 2005, pp. 1541–1546.

[11] E. Tapia, S. Intille, and K. Larson, "Activity recognition in the home using simple and ubiquitous sensors," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, A. Ferscha and F. Mattern, Eds. Springer Berlin / Heidelberg, 2004, pp. 158–175.

[12] J. A. Ward, P. Lukowicz, G. Tröster, and T. E. Starner, "Activity recognition of assembly tasks using body-worn microphones and accelerometers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 1553–1567, 2006.

[13] D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Förster, G. Tröster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, J. Doppler, C. Holzmann, M. Kurz, G. Holl, R. Chavarriaga, M. Creatura, and J. del R. Millán, "Collecting complex activity data sets in highly rich networked sensor environments," in *Proceedings of the Seventh International Conference on Networked Sensing Systems (INSS), Kassel, Germany*. IEEE Computer Society Press, June 2010.

[14] M. Kurz, G. Hölzl, A. Ferscha, A. Calatroni, D. Roggen, and G. Troester, "Real-time transfer and evaluation of activity recognition capabilities in an opportunistic system," in *Third International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE2011), September 25-30, Rome, Italy*, September 2011, pp. 73–78.

[15] M. Botts and A. Robin, "OpenGIS Sensor Model Language (SensorML) Implementation Specification," OGC, Tech. Rep., Jul. 2007.

[16] M. Kurz, G. Hölzl, A. Ferscha, H. Sagha, J. del R. Millán, and R. Chavarriaga, "Dynamic quantification of activity recognition capabilities in opportunistic systems," in *Fourth Conference on Context Awareness for Proactive Systems: CAPS2011, 15-16 May 2011, Budapest, Hungary*, May 2011.

[17] D. Roggen, S. Magnenat, M. Waibel, and G. Troster, "Wearable computing," *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 83–95, june 2011.

[18] G. Hölzl, M. Kurz, P. Halbmayer, J. Erhart, M. Matscheko, A. Ferscha, S. Eisl, and J. Kaltenleithner, "Locomotion@location: When the rubber hits the road," in *The 9th International Conference on Autonomic Computing (ICAC2012)*, September 2012, p. 5.

[19] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 3, pp. 226–239, 1998.

[20] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

 ISBN: 978-1-61208-274-5

Multilevel Planning for Self-Optimizing Mechatronic Systems

Christoph Rasche
C-LAB
University of Paderborn
crasche@c-lab.de

Steffen Ziegert
Department of Computer Science
University of Paderborn
steffen.ziegert@uni-paderborn.de

***Abstract*—This paper presents a multilevel planning approach for the use in complex self-optimizing mechatronic systems. The approach has been designed for the RailCab system, which is an autonomous railbound transportation system. The term multilevel planning denotes planning on different levels of abstraction where each level involves different aspects and thus raises different planning tasks to solve. These planning tasks are not independent of each other. Plans for the higher level planning tasks involve reconfigurations of the system's software architecture and influence parameters used by the lower level planner to compute Pareto optimal behavior. Additionally, the higher level planner relies on plans computed by the lower level planner in order to meet the execution times (of system reconfigurations) that it assumed. To actually assure that the lower level planner computes Pareto optimal plans and takes multiple objectives (which are conflictive) into account, it is based on multi-objective optimization (with Pareto fronts as output).**

***Keywords*—hybrid planning; graph transformation; temporal planning; Pareto front; optimal planning**

I. INTRODUCTION

The ever increasing complexity of mechatronic systems and the integration of more and more sophisticated functionality leads to new challenges for their design and development. The Collaborative Research Centre 614 "Self-optimizing Concepts and Structures in Mechanical Engineering" (CRC 614) at the University of Paderborn treats problems that occur during the design of complex mechatronic systems. The goal is to design self-optimizing systems, that are able to react autonomously to environmental changes by changing their parameters, as well as their objectives, if necessary. The developed concepts go far beyond simple control strategies.

To be able to test the approaches under real world conditions the RailCab system [1], an innovative autonomous railbound transportation systems has been built at the University of Paderborn. It consists of single RailCabs for the driverless transportation of passengers and goods while each vehicle drives on demand. The RailCabs are not coupled mechanically but convoys can be created in order to decrease energy consumption.

A highly complex system like the RailCab system involves various tasks that need to be achieved during runtime. These tasks involve behavior at different levels of abstraction. Each RailCab has an individual goal, e.g., transporting passengers or goods to a specified target station within a specified time. On a high level of abstraction RailCabs plan their route to the target station. Each route in the railway network is assumed to consist of a number of track segments, leading to a discrete planning task at this level. The choice of route and driving speed depends on the routes of other RailCabs in the system. Thus, this abstraction level includes planning of system reconfigurations, like the establishment or breakup of a convoy of RailCabs. These reconfigurations require precise timing information and affect the software architecture of the system, e.g., a convoy operation requires a communication link between the participating RailCabs.

The planning task on the lower level considers continuous behavior and parameters of a single RailCab that has to be planned according to external requirements. For example, a RailCab has to provide a certain level of driving comfort and must not run out of energy before reaching its destination. The fulfillment of these conflictive requirements depends on a variety of control parameters of the RailCab, for which the planner has to determine Pareto optimal settings. The approach is called hybrid planning because it uses an initial discrete plan and forecasts continuous system behavior by simulation during runtime.

Decisions on the higher level, like joining or leaving a convoy, also affect the lower level planning task: due to changing operating conditions, the parameter settings of the lower level planner have to be adapted online to still move with Pareto optimal settings. Since, the planning technique on the lower level refines the higher level actions into continuous behavior, it can guarantee that the next track segment is reached in exactly the time that the higher level temporal planner assumed. Thus, replanning on the lower level will not cause the higher level plan to be invalid afterwards.

This paper presents a hierarchical planning system for self-optimizing mechatronic systems. The temporal planning technique deployed on the higher level of abstraction has been published before in [2], but was treated in isolation only. Here, we specifically address the interrelation of this planning technique with a lower level planner and its ability to adapt its parameter settings online.

The paper is structured as follows. The temporal planning technique, which is used on the higher level of abstraction for planning routes and cooperative behavior, is introduced in Section II. Afterwards, the hybrid planning approach, which is based on multi-objective optimization and used as the lower level planning technique, is described briefly in Section III. The ability to adapt the environmental changes, like emerging drag or temperature changes during movement, is outlined in

 ISBN: 978-1-61208-274-5

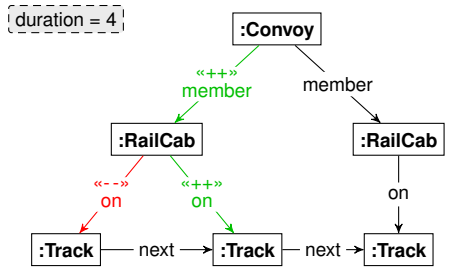


Fig. 1: Story pattern `joinConvoy`

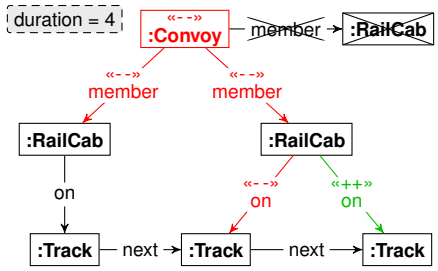


Fig. 2: Story pattern `breakConvoy`

Section IV. The remainder of the paper shows related work in Section V before giving a conclusion in Section VI.

II. Temporal Planning of Software Architecture Reconfiguration

We employ a model-based approach to the design of self-optimizing mechatronic systems. System models are given in MechatronicUML [3], a UML profile for the development of such systems. In the model for the RailCab scenario, the railway system consists of track segments that are connected to each other via `next` links. A RailCab that operates in the system can occupy such a track segment. Furthermore, RailCabs can coordinate with other RailCabs to form a convoy. To safely operate in a convoy, acceleration and braking has to be coordinated and managed between convoy members. Such an active convoy operation is represented by an instance of the `Convoy` type. A `Convoy` instance has a `member` link to each participating RailCab.

This approach to temporal planning deals with software architecture reconfiguration of self-optimizing systems. The communication behavior of components is not considered by this technique. Reconfigurations concerning the software architecture, e.g., the instantiation of a convoy, are modeled with story patterns [4], an extension of UML object diagrams.

Story patterns have a formal semantics based on (typed) graph transformation systems [5]. A graph transformation system consists of a graph representing the initial configuration of the system and a set of rules. Each rule consists of a pair of graphs, called left-hand side (LHS) and right-hand side (RHS), that schematically define how the graph representing the system's configuration can be transformed into new configurations. Elements that are specified in both graphs are preserved, other elements are deleted (if specified in the LHS only) or created (if specified in the RHS only). Syntactically, a story pattern represents such a rule by integrating the LHS and RHS into one graph and using stereotypes to indicate elements that are only present in the LHS or in the RHS.

Fig. 1 provides an example of a reconfiguration, which takes 4 time units: a RailCab joining a convoy of RailCabs. Objects and links that are being created or deleted by the application of the story pattern are labelled with the stereotypes «++» and «--», respectively. The story pattern specifies the creation of a `member` link representing the RailCab's participation in the convoy operation simultaneously with its movement to the next track segment. The story pattern can be executed to transform the state graph into a new configuration if it contains a subgraph that *matches* the LHS of the story pattern.

Our modeling formalism also allows to express that certain objects or links are not permitted to appear in the current state graph. See for example the story pattern given in Fig. 2. The crossed out `RailCab` object and the link connecting it to the `Convoy` object are not allowed to appear in the state graph. Such a restriction to the applicability of a story pattern is called a negative application condition (NAC).

In addition to the story patterns that define possible transformations, we need an initial configuration and a goal specification to feed the planning system with. A goal specification is a partly specified configuration that can be modeled as an ordinary object graph. Goal specifications are either generated from user input or predefined by the system designer. Initial configurations for the planning system are generated from actual runtime states of the system.

Consequently, we are interested not only in which graph transformations to execute but also in the points in time when a graph transformation is supposed to start. In this temporal planning approach, a plan is therefore a set of tuples of points in time and graph transformations. The graph transformations itself have annotated durations.

We solve these planning tasks by translating the models into the Planning Domain Definition Language (PDDL) [6] and feeding them into an off-the-shelf planning system, like SGPlan_6 [7]. In PDDL, a domain is defined by action schemata, as well as types and predicates that can be used within action schemata. An action schema consists of a list of parameters, a precondition, and an effect. In the precondition, a list of literals that are required for applying the action can be specified. Similarly, the effect of an action specifies a list of literals that are obtained when the action is applied. An action is instantiated – in the context of PDDL this is called *grounding* – by substituting the parameters with existing objects. Our translation scheme builds the declarations (of types and predicates) from the class diagram and generates an action schema for each story patterns.

In PDDL, action schemata for time-consuming actions split the literals used in their precondition and effect into different sets according to their time of evaluation. Literals can be required `at_start`, `over_all`, and `at_end` when used in the precondition and be effective `at_start` and `at_end` when used in the effect. The obvious approach to assume that the applicability check happens (in zero time) at the beginning of the reconfiguration and the actual change at its end is not suitable for many situations. Unintended interferences, e.g., the deinstantiation and use of a software component at the same time, could occur. The planning domain has to be generated in a way such that conflicts due to a concurrent execution of

 ISBN: 978-1-61208-274-5

```
60.041: (MOVE rc0 t16 t17) [4.0000]
60.042: (MOVECONVOY convoy0 rc2 rc3 t18 t19 t20) [4.0000]
64.043: (BREAKCONVOY convoy0 rc2 rc3 t19 t20 t21) [4.0000]
64.044: (MOVE rc0 t17 t18) [4.0000]
68.045: (MOVE rc3 t21 t22) [4.0000]
68.046: (CREATECONVOY convoy0 rc2 rc1 t19 t25 t26) [4.0000]
72.047: (MOVE rc3 t22 t23) [4.0000]
72.048: (MOVECONVOY convoy0 rc2 rc1 t25 t26 t27) [4.0000]
72.049: (MOVE rc0 t18 t19) [4.0000]
```

Fig. 3: Excerpt of a concurrent reconfiguration plan

reconfigurations are not possible. However, we do not want to burden the designer of the planning domain with its complicated and error-prone definition. Therefore, the questions that our translation scheme needs to address are: does the concurrent execution of two graph transformations result in any conflicts, and how can such a concurrent execution be avoided? To safely control whether a concurrent execution is allowed, our solution generates additional literals that *lock* access to graph nodes and edges when they are in use by a reconfiguration.

Consider for instance the application of the story patterns `joinConvoy` and `breakConvoy` given in Fig. 1 and 2. Let us assume that one of the reconfigurations, e.g., `breakConvoy`, is currently being applied. This means, its condition has already been checked but the alteration of the configuration has not yet been executed. The execution of a reconfiguration of RailCab `r1` joining the convoy makes no sense in this situation and should not be allowed because the convoy will be deinstantiated by `breakConvoy`. The problem is that the configuration is in the process of being changed, but this is not reflected in the intermediate state graph. Checking the applicability at the beginning of a reconfiguration and executing the alteration at its end is ineligible as a general solution. Our solution to this problem encodes information about the deinstantiation of the convoy into the configuration by acquiring a write lock of the `Convoy` object when the `breakConvoy` reconfiguration starts and releasing the lock when the reconfiguration ends. In the opposite case, i.e., if `joinConvoy` starts first, it encodes into the configuration that it requires the `Convoy` object by acquiring a read lock and releasing it when the reconfiguration ends. This approach is very suitable for a translation into PDDL since locking functionality can simply be realized by defining new predicates and functions for the locks. Since acquiring and releasing all locking literals of a reconfiguration is done as an atomic step (at the beginning and the end of the reconfiguration, respectively), there can be no deadlocks when acquiring the locks. For more details on the translation scheme we refer the reader to [2].

Our model includes story patterns to move RailCabs or convoys of RailCabs and story patterns related to convoy de-/instantiation and membership change. All these story patterns are available in the generated planning domain as action schemata.

Listing 3 shows an excerpt of a plan that was generated by SGPlan_6 for a planning task involving 4 RailCabs. During the interval [60–64], RailCabs `rc2` and `rc3` operate in convoy mode. From 64 to 68, they break up the convoy operation because the underlying domain specifies a Y junction between tracks `t19`, `t20`, and `t25`, and they need to move along different routes to arrive at their target locations. To do so, `rc2` has to fall back, i.e., it still occupies `t19` at 68. Concurrently, i.e., during the interval [60–68], `rc0` moves from `t16` to `t18` but waits from 68 to 72 to not crash into `rc2`. To sum up, the reconfiguration plans that SGPlan_6 produces take advantage of parallel execution of actions when possible, while guaranteeing that concurrently executed actions do not interfere with each other. With regard to the application scenario, this means that RailCabs operate in parallel if they are sufficiently apart from each other, but wait for the execution of other RailCabs' reconfigurations if necessary, e.g., to clear a common track segment.

III. Hybrid Planning

Besides the temporal planning approach for the coordination of several RailCabs, e.g., to create a convoy each RailCab computes single behavioral plans in order to move from its initial position to its destination with Pareto optimal settings on the lower level. Such plans are necessary as the RailCab has only limited energy resources. The passengers can, e.g., change the velocity of their RailCab, which influences the energy consumption.

A hierarchical hybrid planner is used for the purpose of creating behavioral plans [8]. This hybrid planning approach first computes an initial plan and forecasts continuous system behavior. The purpose is the creation and adaptation of a plan in order to always move with Pareto optimal settings from the start position to the destination. The problem of planning is modeled as a linear optimization problem and solved using the Simplex algorithm.

Such a plan has to consider conflictive objectives. These objectives have to be optimized which leads to a multi-objective optimization problem. A Pareto set is the solution set of such a problem and forms a $(k-1)$-dimensional object. In this context, k denotes the number of objectives involved in the problem. The computation of a single Pareto front for each track section is neglected in this paper. The interested reader is referred to a detailed description in [10].

A multi-objective optimization, using the program GAIO [9], [10] is performed, which results in Pareto fronts. Based on the Pareto fronts several Pareto optimal configurations of system parameters with different degrees of performance regarding each objective are provided to the planner. The planner then selects one of these sets of Pareto points for each track section. A plan then consists of a set of Pareto points leading to Pareto optimal settings at each track section.

The initially computed plan, based on the Pareto points computed before starting the journey, does not take into account the actual operating conditions during the journey, which might deviate from the a priory assumed conditions, e.g., due to environmental changes. Hence, only preliminary parameters can be taken into account for the computation of feasible plans.

Creating and dissolving convoys, for example, leads to states where the RailCabs do not move any longer with Pareto optimal settings due to a considerable change of drag. In a convoy each RailCab behind the leader moves in the slipstream of the RailCab in front of it. This changes the energy consumption at consistent movement speed dramatically for most of the RailCabs and a new plan considering the new energy consumption should be computed for every RailCab

 ISBN: 978-1-61208-274-5

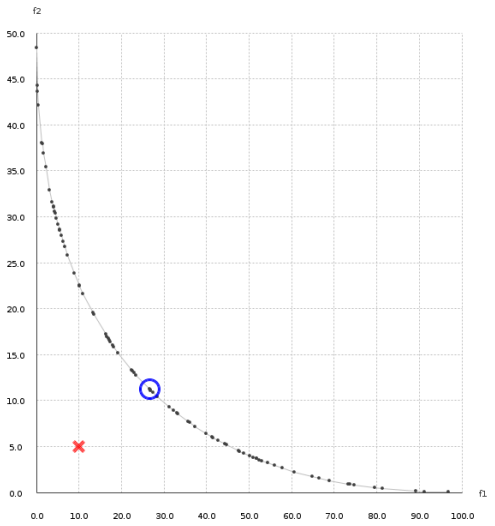


Fig. 4: An example Pareto front. The blue circle represents the Pareto point selected by the planner while the red cross shows the working point.

that is affected by these changes. Additionally, the energy consumption has to be decreased when moving in a slipstream as the current energy consumption would increase the speed of the RailCab and the timing assumptions, on which the temporal planner relies, would not be met.

The same is true, if a RailCab leaves a convoy. It then faces a higher drag and thus needs more energy to keep its speed constant. Therefore, an adaptation of the Pareto front is performed, as described in the following section. The planner then computes a new plan that is based on new parameters, which are calculated by the adaptation approach, in order to always work on valid data.

IV. Behavioral Adaptation of Differing Model Parameters

As mentioned, it is hardly possible to avoid deviations between the settings achieved by the use of previously computed Pareto fronts and the real settings, reached during operation. From this it follows that the Pareto points, on which the original plan is based, become invalid and thus the entire plan becomes invalid. To still be able to use a plan, which is as close to be optimal as possible, a change of the Pareto front has to be conducted.

Fig. 4 shows an example for a parameter change. The line denotes the Pareto front computed by the use of the model values and the blue circle shows the Pareto point selected by the planner. Each Pareto point considered in this paper has a comfort value, given by f_1 and an energy consumption value, given by f_2. The currently measured comfort and energy consumption leads to the working point denoted by a red cross. This example includes considerable changes in the energy consumption as well as between the measured comfort value and the comfort value given by the selected Pareto point. Such differences make the entire plan invalid and a recalculation, based on the newly determined working point must be conducted.

To be able to detect such deviations and to change the plan, several values like energy consumption and passenger comfort have to be measured continuously during RailCab operation. Based on the measured values the current working point has to be calculated.

It is not possible to simply compute new Pareto fronts, leading to the measured working point as the changes of the environmental parameters are not computable and it is possible that the current working point is not Pareto optimal. In that case no Pareto front, containing the working point exists. Thus, an iterative approximation of the model Pareto front towards the measurements is conducted.

A. Taylor Series Approximation

An approximation using Taylor Series expansion can be performed, to successively approximate a Pareto front given by model parameters closer to a working point, obtained by measured values [11].

The functions, used to compute Pareto fronts for an entire RailCab are rather complex and mainly a combination of the multi-objective functions for single systems, presented in [8], [12], [13] and [14]. The resulting Pareto points selected by the planner are n-dimensional. To adapt a Pareto point to a working point each dimension is considered individually.

Let x_0 be the Pareto point and let x_m be the working point. First, a Taylor series for each parameter p_i in dimension i of the multi-objective function is computed, using the partial derivatives as follows:

$$T_i(x) = f(x_0) + \frac{\frac{\partial f(x_0)}{\partial x_i}}{1!}(x - x_0) + \frac{\frac{\partial^2 f(x_0)}{\partial x_i}}{2!}(x - x_0)^2 + \frac{\frac{\partial^3 f(x_0)}{\partial x_i}}{3!}(x - x_0)^3 + \cdots \quad (1)$$

Additionally, the differences between the Pareto point and the working point for each parameter $\Delta p_i = \|x_0 - x_m\|_i$ are computed. These differences are used to compute new parameter values p_{i_n}, as shown in the following equation.

$$p_{i_n} = p_i + \frac{\Delta p_i}{T_i(x_m)} \quad (2)$$

The Pareto front is then newly computed, based on this new parameters and the procedure is started over again until no further reduction of the differences is achieved. The resulting Pareto front is close to the working point and based on the new environmental parameters. The planner then uses Pareto fronts, based on the newly determined parameters to conduct a replanning, leading to a new and feasible path.

B. Result

Fig. 5 shows an example for a recalculated Pareto front. The values used in this example are abstract values without units of measurement. Two simple functions were used to represent passenger comfort and an energy consumption at a specific movement speed. The functions lead to a two-dimensional Pareto front. The axis f_1 depicts the current energy consumption and the axis f_2 the current passenger comfort.

 ISBN: 978-1-61208-274-5

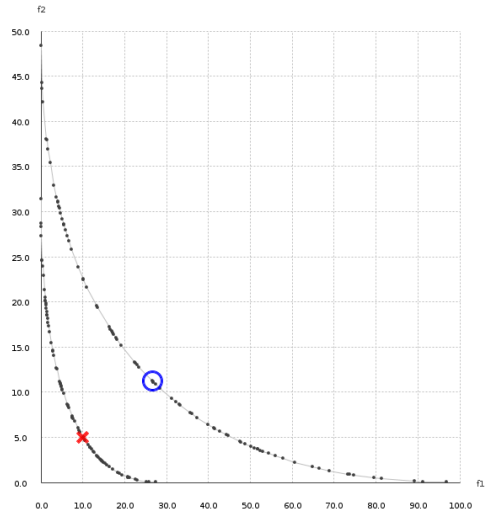


Fig. 5: Recalculated Pareto front based on measurement point. The blue circle shows the Pareto point selected by the planner. The red cross denotes the working point. The original Pareto front has been adapted towards the working point.

A low comfort value results in a high passenger comfort. In the depicted example scenario the planner selected the Pareto point at position (26.5,11.2) from the model Pareto front for the current track section, framed by a blue circle in order to reach the required comfort without consuming too much energy. The measured data revealed a current working point at position (10,5), which is framed by a red cross.

Such a difference between the Pareto point and the working point makes the overall plan, used to move to the destination invalid. In order to be able to compute a new plan an adaptation of the Pareto front towards the working point is conducted, as shown in Fig. 5. Based on these resulting values new Pareto fronts can be computed, used by the planner to create a feasible plan.

V. Related Work

The multilevel planning approach presented in this paper is based on the assumption that the planning tasks are inherently hierarchical and can be separated into higher level and lower level tasks. No interleaving between the planning techniques is necessary; thus, they can be applied sequentially on their respective abstraction levels (beginning at the highest level). For systems where a strict separation of the planning tasks is not possible, Marthi et al. [15] proposed an approach called *angelic* hierarchical planning. Their approach provides the higher level planner with models that allow to make guarantees on the satisfiability of the lower level planning tasks. This prevents the system from having to backtrack to a higher abstraction level, which results in a significant speedup.

When researchers try to tie AI planning techniques with the software engineering domain, their techniques often rely on graph transformation systems. Estler and Wehrheim [16] developed a heuristic search planner along with a technique to learn domain-specific heuristics from modeling artifacts (among other things from a meta-model). These techniques are promising because of their intuitive representation and close association to model-based software engineering. However, up to today, they usually do not support time-consuming reconfigurations.

Similar to our approach for the generation of temporal plans, Tichy and Klöpper [17] presented an automatic translation of graph transformation rules into PDDL actions to plan self-adaptive behavior. The support for time-consuming reconfigurations was addressed only in terms of stereotypes; concurrency issues were not treated. Our temporal planning technique can be seen as an extension of their approach.

The use of Pareto optimal solutions is a common approach for planning in multi-objective applications. Hongfu et al. [18], e.g., solve the multi-objective problem of intelligent mission planning in dynamic environments by a combination of Pareto fronts, receding horizon control, fuzzy inference systems and expert knowledge. In contrast to the approach presented in this paper they select a Pareto point from the computed front, based on expert knowledge.

Klöpper et al. [19] use Pareto based planning in multi-agent mechatronic systems. In their system operation strategies exist, that correspond to Pareto optimal configurations. These operation strategies represent trade-offs between the system objectives and expected system state changes regarding limited resources, as e.g., energy and execution time. The resulting Pareto optimal configurations are used as input to the planning model, which is based on a state-action formalism. They also use a hybrid planning approach, combining local planning and reactive behavior for decision making in real-time.

VI. Conclusion

We presented a multilevel planning approach for self-optimizing mechatronic systems that adapt itself to envoronmental changes. Many other planning approaches do not consider environmental influences. A plan is often computed a priori and followed by the autonomous system. Changes of the environment, like newly detected obstacles, can then force a replanning. Nevertheless, most times it is assumed that parts of the environment, like weather, temperature, etc., have no influence to the system. There are several applications for which such an assumption does not hold. The presented RailCab system is such a system that is directly influenced, e.g., by drag changes which influences the energy consumption.

In our approach, Pareto optimal plans cannot be created a priory due to unpredictable environmental influences, like emerging headwind or temperature changes during movement. Such influences lead to changing parameter values that have to be taken into account. Therefore, the Pareto front that the initial plan is based on is adapted online towards a measured working point. After such an adaptation, the planner replans using the updated parameter values.

In addition, there are planning tasks on a higher abstraction level that are solved by an independent planning system. Plans for the higher level planning tasks involve reconfigurations of the system's software architecture and influence parameters used by the lower level planner to compute Pareto optimal behavior. This higher level temporal planner allows to execute system reconfigurations in parallel. Its planning tasks are solved by translating them into a PDDL representation that can be handled by off-the-shelf planning systems.

 ISBN: 978-1-61208-274-5

Our combined approach is a first step towards a self-optimizing system that computes temporal reconfiguration plans, which change the software architecture of the system and solve conflictive objectives. Furthermore, it is able to adapt its control parameters to environmental changes like changing drag. In theory, it can be applied to several applications that need to take temporal properties and environmental changes into account.

ACKNOWLEDGMENT

This work was developed in the course of the Collaborative Research Centre 614 “Self-optimizing Concepts and Structures in Mechanical Engineering” and funded by the German Research Foundation (DFG).

REFERENCES

[1] C. Henke, M. Tichy, T. Schneider, J. Bocker, and W. Schafer, “System architecture and risk management for autonomous railway convoys,” in Systems Conference, 2008 2nd Annual IEEE, April 2008, pp. 1–8.

[2] S. Ziegert and H. Wehrheim, “Temporal reconfiguration plans for self-adaptive systems,” in Software Engineering (SE 2013), ser. Lecture Notes in Informatics (LNI). Gesellschaft für Informatik e.V. (GI), February 2013.

[3] S. Becker et al., “The MechatronicUML design method – process, syntax, and semantics,” Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, Tech. Rep., 2012.

[4] T. Fischer, J. Niere, L. Torunski, and A. Zündorf, “Story diagrams: A new graph rewrite language based on the unified modeling language,” in 6th Int. Workshop on Theory and Application of Graph Transformations (TAGT 1998), 1998.

[5] H. Ehrig et al., “Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach,” in Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations, G. Rozenberg, Ed. World Scientific, 1997, ch. 4, pp. 247–312.

[6] M. Fox and D. Long, “PDDL2.1: An extension to PDDL for expressing temporal planning domains,” Journal of Artificial Intelligence Research (JAIR), vol. 20, 2003, pp. 61–124.

[7] Y. Chen, B. W. Wah, and C.-W. Hsu, “Temporal planning using subgoal partitioning and resolution in SGPlan,” Journal of Artificial Intelligence Research (JAIR), vol. 26, 2006, pp. 323–369.

[8] N. Esau et al., “Hierarchical hybrid planning for a self-optimizing active suspension system,” in 7th IEEE Conference in Industrial Electronics and Applications, IEEE. Singapore: IEEE, 18 - 20 Jul. 2012.

[9] M. Dellnitz, O. Schütze, and T. Hestermeyer, “Covering Pareto sets by multilevel subdivision techniques,” Journal of Optimization Theory and Application, vol. 124 (1), 2005, pp. 113–136.

[10] O. Schütze, K. Witting, S. Ober-Blöbaum, and M. Dellnitz, “Set oriented methods for the numerical treatment of multi-objective optimization problems,” in EVOLVE – A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation, ser. Studies in Computational Intelligence, E. T. et al., Ed. Springer Berlin Heidelberg, 2013, vol. 447, pp. 187–219.

[11] J. Li, H.-C. Zhang, and Z. Lin, “Asymmetric negotiation based collaborative product design for component reuse in disparate products,” Computers & Industrial Engineering, vol. 57, no. 1, 2009, pp. 80–90.

[12] C. Romaus, J. Bocker, K. Witting, A. Seifried, and O. Znamenshchykov, “Optimal energy management for a hybrid energy storage system combining batteries and double layer capacitors,” in Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE, September 2009, pp. 1640–1647.

[13] A. Trachtler, E. Munch, and H. Vocking, “Iterative learning and self-optimization techniques for the innovative railcab-system,” in IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on, November 2006, pp. 4683–4688.

[14] A. Pottharst et al., “Operating point assignment of a linear motor driven vehicle using multiobjective optimization methods,” in Proc. of the 11th International Power Electronics and Motion Control Conference (EPE-PEMC 2004), 2004.

[15] B. Marthi, S. J. Russell, and J. Wolfe, “Angelic hierarchical planning: Optimal and online algorithms,” in Int. Conf. on Automated Planning and Scheduling (ICAPS 2008), 2008.

[16] H.-C. Estler and H. Wehrheim, “Heuristic search-based planning for graph transformation systems,” in Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2011), 2011, pp. 54–61.

[17] M. Tichy and B. Klöpper, “Planning self-adaptation with graph transformations,” in Int. Symp. on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2011), 2011.

[18] H. Liu, X. Gu, J. Chen, and H. Liu, “Intelligent multi-objective receding horizon control for ucav mission planning,” in Computer Science and Information Processing (CSIP), 2012 International Conference on, August 2012, pp. 1154–1158.

[19] B. Klöpper, S. Honiden, and W. Dangelmaier, “Divide & conquer in planning for self-optimizing mechatronic systems - a first application example,” in Computational Intelligence in Control and Automation (CICA), 2011 IEEE Symposium on, April 2011, pp. 108–115.

 ISBN: 978-1-61208-274-5

Testing the Reconfiguration of Adaptive Systems

Kai Nehring, Peter Liggesmeyer
AG Software Engineering: Dependability
University of Kaiserslautern
Kaiserslautern, Germany
Email: nehring@cs.uni-kl.de, liggesmeyer@cs.uni-kl.de

***Abstract*—Adaptive systems can change their internal structure in order to respond to changes in their environment. These changes can cause malfunctions if not applied correctly. Current test approaches do not cover every aspect of the reconfiguration sufficiently. In this paper, we present a novel approach for testing the reconfiguration process of adaptive systems with respect to structural changes. The approach presents testers with guidelines for choosing the appropriate test strategy for a certain aspect, such as order of reconfiguration, state transfer, and transaction handling, of the reconfiguration procedure.**

Keywords-adaptive system; testing; reconfiguration; test process model; structural changes

I. Introduction

Dynamically adaptive systems are a promising alternative to static systems when a system must modify its structure or behaviour due to changes in its executional context. Testing such systems, however, can be a challenging task since their structure and even their functionality may change at runtime — a test approach would have to take this into account. Current test- and verification approaches focus on the functionality and execution environment[4], the adaption policy[5][7], and automated evaluation of structural changes[3], but not on the reconfiguration itself on the executable system. Furthermore, some test approaches require complex (formal) models of the system in order to be applicable[6].

We have already illustrated how the visualisation and inspection of structural changes in adaptive systems can help to detect potential defects, and, at the same time, omit formal models[1]. However, other aspects, such as the state transfer from an object to its replacement, which are often interwoven with structural changes remained uncovered, too. We propose a novel approach to test the reconfiguration process of adaptive systems with respect to structural changes. The approach is designed as an additional test activity, hence it is a supplement and not a replacement for the aforementioned approaches.

In Section II, we propose a process model to test the reconfiguration procedure. Section III summarises our experience when we applied the test process on adaptive systems. Section IV completes with a conclusion and an overview of the future work.

II. Test Process Model

We propose a self-contained iterative test process model that addresses issues that may arise when structural changes are involved during the reconfiguration, such as replacing the instance of component X with an instance of component Y. The test process model is comprised of 6 iterations, each of which focuses on a specific aspect of the reconfiguration, such as the state transfer between a component and its replacement. Furthermore, the process model can be tailored to fit the system under test, i.e., iterations can be omitted if they focus on an aspect that is not supported in the system under test. Each iteration is further divided into three phases:

1) **Preparation**, in which workload, instrumentation probes, etc. will be prepared
2) **Execution**, in which the workload will be executed in order to collect data about the system's behaviour
3) **Evaluation**, in which the obtained runtime data is evaluated

Breaking down an iteration into phases not only allows the reuse of information from other iterations, but also to split and dispatch the procedure to multiple roles. The **Domain expert** provides the workload (i.e., the input data) that is executed at runtime. The expert also evaluates the result of the processing and determines whether the system processes the data correctly. **Developer** and **System Manager** execute the workload on the instrumented system. The **Architect** evaluates the structural changes.

The separation into phases has been omitted for simplicity reasons in the following outlines of the iterations.

A. Iteration 1 — System Overview

Understanding the internal structure of a system is essential to estimate effects of changes in its structure. Design documents are often a valuable source of knowledge, but they are, however, not always very reliable for several reasons:

- Development and maintenance can cause the system's structure to change over time. Changes in the source code are not always reflected in the design documents.
- The system has not been implemented as specified.
- etc.

 ISBN: 978-1-61208-274-5

Goal of Iteration 1 is to create a (virtually) complete runtime model of the system, which can later be used to select the actual components for further investigation. However, the iteration can be omitted if detailed and reliable knowledge of the system's runtime composition is available.

Assignments to attributes, which hold references to other objects and their values, must be tracked in order to create a runtime model. Since design documents can be outdated, inconsistent, or incomplete, the attributes are best collected from the source code. Instrumentation probes must be created for each attribute to track changes. Furthermore, a representative workload must be prepared in order to execute the system's functionality in different states/configurations, and to cause reconfigurations. An external trigger must be prepared if the reconfiguration is not induced by the workload itself.

The workload must be executed on the instrumented system. Depending on the approach that is used to implement the functionality, the workload might be required to run twice — before and after the reconfiguration — to track all utilised object–instances. Particularly components which utilise lazy loading would be incomplete otherwise.

Analysis of the collected information will not only unveil the object/component composition at runtime but also the order of changes during the reconfiguration. However, the vast amount of information may be overwhelming and may make further analysis more difficult since potentially lots of objects are displayed although they are not linked to the reconfiguration. Furthermore, the runtime behaviour of the system might be altered due to the instrumentation overhead, which may cause the system to change the reconfiguration strategy[2].

B. Iteration 2 — Structural Changes

The order of instructions required to perform a reconfiguration is usually flexible to some degree. Although all considered execution paths eventually lead to a valid composition or configuration, quality-of-service and system integrity might be affected by a particular strategy. A reconfiguration strategy which focuses on system integrity might passivate all components before changes are performed — this might result in a reduced quality-of-service since the system is either unavailable or operates in a gracefully degraded mode (for a longer period of time). A quality-of-service based approach might try to minimise the downtime by performing as many steps as possible parallel to normal operation, which may increase the risk of inconsistencies in the system's data. The structural changes in the course of the reconfiguration are evaluated in Iteration 2.

Typically, only few components are affected by a reconfiguration. Tracing changes in such components can help in breaking up the complex system into partitions. The result of Iteration 1 can be used to eliminate unnecessary components, which results in a smaller set of probes to instrument the system. A component can be considered *unnecessary*, if it is neither involved in, nor affected by the reconfiguration.

The workload must fulfil the same requirements as in Iteration 1, and it can even be reused if Iteration 1 has been executed. It must then be executed on the instrumented system.

Analysis of the trace is best done using a graphical representation, such as a series of object diagrams[1]. In such an approach, the trace will be transformed into a series of object diagrams, at which each state, caused by a change in the structure, is expressed by a new object diagram. Developers and architects evaluate each diagram and decide whether the system passed through an illegal state. Unlike other methods, such as AMOEBA-RT[3], this approach does not require a formal model, such as a temporal logic model, to describe the system states.

C. Iteration 3 — Data Integrity

The system is in a transitional state during the reconfiguration. It can operate with either reduced or mixed functionality, be non-functional at all, or, in the worst case, in an inconsistent operational state. The approach used to achieve adaptability not only has a great influence on the observable behaviour, but also on system integrity and data integrity. Iteration 3 tests the data integrity in presence of a (increasing) load, such as incoming user requests.

The type of workload depends on the strategy used to achieve adaptability, and is distinguished between step load and ramp load. While *ramp load* slowly increases the workload on the system, *step load* suddenly puts a lot of pressure on the system, displayed in Figure 1. The system's behaviour on step load can point to potential defects, if the system implements a load-based adaptation strategy[2]. Furthermore, if the system is designed to buffer user requests while it is reconfigured, then a step load-like increase of requests could cause the buffer to run out of space before the reconfiguration has been completed. To test the behaviour of the system under these conditions, the load must be sized accordingly.

The system should not show unexpected behaviour, such as crashes, or lost requests, that can be traced back to the reconfiguration. Furthermore, it is advisable to verify that all structural changes during the reconfiguration have been applied correctly in order to preclude the possibility that a faulty trigger prevented the reconfiguration. If a queue (or, in more general, a "buffer") is used to buffer user requests, additional tests are required to ensure that it satisfies the following requirements:

- The queue/buffer must either be properly sized or be resizable in order to prevent the loss of user requests according to the quality requirements.
- The order of requests must be preserved, if not otherwise specified.

 ISBN: 978-1-61208-274-5

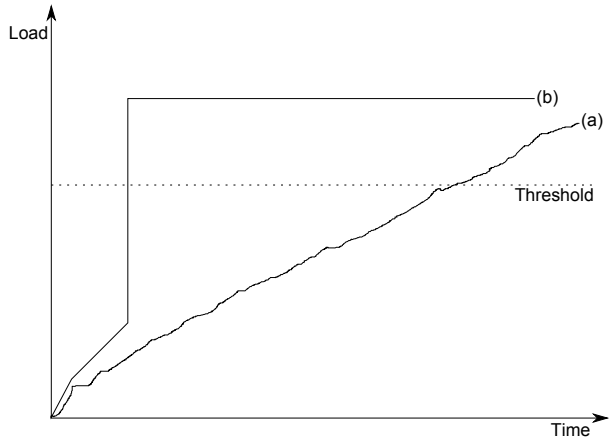


Figure 1. Ramp load vs. step load: ramp load (a) changes continuously whereas step load (b) changes suddenly. Reconfiguration of the system occurs if the load exceeds a predefined threshold.

- All buffered requests must be processed after the reconfiguration has been completed, or actions have to be taken if the reconfiguration failed. The order of the execution must be equal to the order in which the requests were buffered, if not otherwise specified.
- New requests must be buffered until all previously received requests have been executed, if not otherwise specified.

D. Iteration 4 — State Transfer

A stateful component may be instructed to transfer its internal state to the replacement if it is about to be replaced. Iteration 4 checks whether the state of a component will be transferred correctly to its replacement, and, if necessary, type conversion is done in a way so that the replacement is fully operational. Furthermore, potential access violations due to the reconfiguration on concurrent systems can be checked.

A set of instrumentation probes should be prepared in order to monitor the runtime composition. If a component X is about to be replaced with Y, and both X and Y are instances of the same type, then the instrumentation facility must be able to distinguish them, e.g., by recording their memory addresses.

Furthermore, two workloads must be prepared. The first workload is applied on the system before the component is about to be replaced — the preload phase. It is responsible to set up the state of the instance of component X that must then be transferred to the replacement Y. The second workload must be executed either during or after the reconfiguration, depending on whether the system utilises concurrency.

On a purely single threaded system, the second workload is executed solely to verify that the new instance Y is fully operational and (optional required) conversion of the internal state has been successfully carried out.

On a multi threaded system, the second workload must be executed while the reconfiguration is running to test the following situations:

- The system must not alter the state of instance X once the state transfer has begun, i.e., user requests have to be stored in a buffer, if not otherwise specified.
- The system must be fully operational after the reconfiguration, i.e., the state transfer has been successfully carried out.

After the state transfer (of instance X) has been performed and workload 2 has been executed, the replacement (instance Y) should be comprised of the required data of instance X and of the new data. Depending on the type of system, this can manifest in the following situations:

- All data, e.g., items in a shopping trolley, have been added to the replacement.
- All data, e.g., GPS coordinates of a route, have been added to the replacement and the order has been preserved. In addition, all new data have been appended to the previous ones.
- A new state, e.g., a new random number, has been calculated correctly using the previous state of instance X.

E. Iteration 5 — Transaction Handling

The reconfiguration of a system can impact transaction capable components either directly, if the component is target of a reconfiguration, or indirectly, if the transaction capable component utilises a component that is part of the reconfiguration or vice versa. Iteration 5 checks the transaction handling during a reconfiguration.

The system specification should provide information about the observable behaviour when a reconfiguration is triggered while a transaction is running. It can most likely be narrowed down to the following two situations:

1) The reconfiguration must be delayed until the transaction has been finished. A finished transaction can be either successful or unsuccessful in which case rollback has to be performed.
2) The transaction must be aborted and a rollback must be performed to undo changes. Nevertheless, the reconfiguration must be delayed until the rollback has been completed to ensure data integrity.

The progress of a transaction is, however, of no interest, i.e., a component X must not be passivated (e.g., in order to replace it) even if its job is done. In case of a rollback, which can still occur if the last operation in the transaction fails, the component X might be needed again.

A set of instrumentation probes should be prepared in order to monitor the component composition. Also a workload must be prepared to utilise the system. Furthermore, the reconfiguration must be triggered while a transaction is running.

 ISBN: 978-1-61208-274-5

During the execution, a snapshot of the datasource that is about to be altered should be made to simplify the integrity check after the reconfiguration has been performed. The reconfiguration must not only be triggered while the workload is executing but also while a transaction is running.

The evaluation of the reconfiguration encompasses several steps. First, the system composition must be checked — the system must be in an operational state. This includes that no crashes or deadlocks occurred at runtime due to an incomplete system composition (e.g., crashes due to null pointer exceptions) or passivated components.

Depending on the specification of the system, the transaction must be either be rolled back or the reconfiguration must be delayed until the transaction has been finished. The latter includes a rollback, i.e., the system must not change its composition while a transaction is running.

In the last step, the data integrity must be evaluated. The data must either be unaltered (in case of a rollback) or fully updated.

F. Iteration 6 — Identity

It is in some cases important to know whether one or more components use the same instance of a component X or instances of the same type T, where X is an implementation or a subtype of T. Iteration 6 focuses on the identity of the instances.

A typical workload must be executed on an instrumented system. The instrumentation facility must be able to distinguish multiple instances of the same type. Common practice is to record the memory address of each instance. Developers then compare the utilised objects and determine whether the usage scenario is acceptable.

III. Evaluation

We have evaluated the test approach on three systems so far:

1) adaptive Tic-Tac-Toe: a version of the well known game Noughts and Crosses, which automatically adjusts the game level of the computer player in relation to the human player's skills
2) an adaptive ERP system, which utilises a local cache if the connection to the off-site master ERP system is faulty. It automatically synchronises and utilises the off-site ERP as soon as it becomes available again
3) an Emergency Detection System, which can be used to monitor humans in an ambient assisted living scenario. The system utilises a variety of sensors, such as pulse sensors, fall sensors, etc. A new sensor will be utilised after it becomes available to the system and if the quality-of-service level can be improved by the new sensor

A. adaptive Tic-Tac-Toe

The only reconfigurable component in the program is the computer player, whose playing strategy (Easy, Medium, and Advanced) can be adjusted after each match according to the following predefined rules:

1) The game level shall be increased to the next higher level if the computer loses two consecutive matches. The game level remains unaltered if the highest game level has already been reached.
2) The game level shall be decreased to the next lower level if either
   a) two consecutive matches end with a draw, or
   b) the computer player wins two consecutive matches.

   The game level remains unaltered if the lowest game level has already been reached.

Applicable iterations are:

- Iteration 2 to create an overview of the reconfiguration process and to track structural changes
- Iteration 6 for an overall view of the used instances

The structure of the game is rather simple since only the play strategy can be varied. In the course of tailoring, the remaining iterations have been removed for the following reasons:

- The structure of the computer player is simple; an complete overview is unnecessary
- The play strategy cannot be replaced while a match is running
- No state transfer among the game strategies is supported
- The game does not utilise transactions

Furthermore, the tracing of the program would deliver identical results for Iteration 2 and Iteration 6, which is why we reused the tracing results of Iteration 2 for further analysis in Iteration 6.

The evaluation of the trace unveiled abnormal behaviour whenever the human player repeatedly won matches in the game level *Advanced*. The computer player reconfigured itself even though it already utilised the *Advanced* strategy level, which contradicts the requirements. Also, we could observe a strange behaviour where the *Advanced* level was replaced with another instance of the same class, which does not change the behaviour of the computer player. Further examinations located the defect in the method which increases the game level. The method did not identify the level *Advanced* correctly. Traditional testing did not uncover the defect since *Advanced* is the highest level in the current version of the game. Furthermore, the game did not expose unusual behaviour to justify further investigations. However, the defect would have caused stagnancy in the level *Advanced* once further levels would have been added to the game.

 ISBN: 978-1-61208-274-5

B. ERP System

The class diagram in Figure 2 shows a simplified overview of the system. The `CashRegisterController` comprises a reference to an implementation of the `ERP` interface. This reference is subject to change if the master ERP is replaced with the cache, and vice versa.

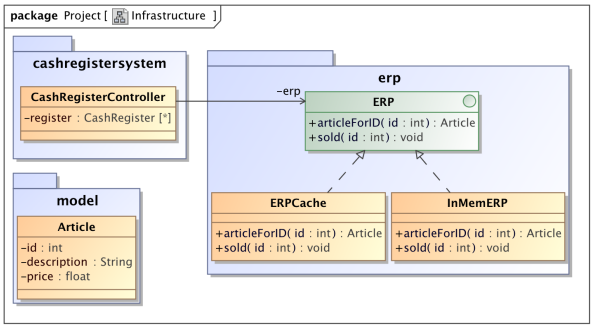


Figure 2. ERP class diagram (Key: UML class diagram)

A full system trace in not required since the reconfigurable portion of the system is rather small. The current version does not allow a cash register system to operate while the system is being reconfigured, which is why Iteration 3 is not applicable. Neither is Iteration 5 since the system does not utilise transactional components beyond the database management system. Besides the reconfiguration process itself (Iteration 2) and the state transfer test (Iteration 4), the component identity (Iteration 6) is subject to test to ensure that the same master ERP system is used, i.e, the same server address and identical login credentials are used.

Two sets of data were prepared in order to test the state transfer. The first set was used to preload the system with information, which were stored in the master ERP's database. The second data set was used to populate the ERPCache after the reconfiguration "master ERP to ERPCache". That data was transferred to the master ERP once it became available again, which resulted in a combined set of data.

Two snapshots were created to gather the internal state of the master ERP — one before the first reconfiguration and one after the final reconfiguration. The analysis of the ERP-data did not unveil deviations from the expected data, i.e., the state transfer has been implemented correctly.

The tracing results of Iteration 2 were reused to check whether the master ERP was the same before the first reconfiguration (transition to the ERPCache) and after the second reconfiguration (transition back to the master ERP). The hash codes of the utilised master ERP instances were equal in both cases, i.e., the same ERP system was used.

C. Emergency Detection System

The Emergency Detection System (EDS) is a monitoring system, applicable for example in an ambient assisted living scenario. It constantly evaluates informations which it acquires from several sources, such as blood pressure sensors, pulse sensors, and location sensors. If it detects a critical situation, it can execute a variety of protocols, e.g., notify emergency medical staff. A critical situation can be caused, for example, by a sudden change in vital signs, or by a fall of the monitored person. The system automatically selects a sensor configuration to offer the highest possible service quality. Furthermore, it reconfigures itself to utilise newly added sensors without service interruption.

If a new sensor is registered at the `EDSManager`, which is a centralised administration component to keep track of all available sensors, its contribution to the system is analysed. The system will be reconfigured to utilise the new sensor if the new sensor is considered *valuable*. A sensor is rated valuable if the EDS-evaluation-algorithm can create a sensor configuration that results in a higher quality-of-service, either by adding new kind of sensor, which was not available before (e.g., a fall sensor), or by replacing an existing sensor with a higher grade sensor (e.g., higher Safety Integrity Level (SIL)). The sensors are connected to a an implementation of the `IEDSHandler`-interface. Each supported configuration is represented by its own implementation, i.e., an implementation that supports only a pulse sensor and a pressure sensor can be distinguished from an implementation that supports pulse sensor, pressure sensor, and a location sensor. Each `IEDSHandler`-implementation processes the sensor-signals and sets off the alarm if a critical situation is detected. This design offers flexibility since a new handler can be set up in the background. If it is fully constructed, the new handler replaces the old one without service interruption.

According to the system description, sensor signals are not processed while the handler is replaced, and historic data is not transferred either. Hence, Iteration 3 and Iteration 4 are not applicable. The system does not support transaction, which is why Iteration 5 is not applicable, too. The component identity (Iteration 6) is of interest to identify which sensors are utilised at a particular point in time.

The reconfiguration test had been divided into three stages:

1) The initial system configuration comprises a blood pressure sensor (SIL 2) and a pulse sensor (SIL 2)
2) A location sensor (SIL 1) is added — the system should integrate the location sensor.
3) A new pulse sensor (SIL 3) is added — the system should replace the previously used SIL 2 pulse sensor with the SIL 3 pulse sensor.

Starting from the initial configuration, the location sensor was added. The system incorporated the location sensor without service interruption. Then the new pulse sensor was added. The evaluation of the object diagrams showed that a new handler was created, which utilised the pressure sensor, the location sensor, and the new (SIL 3) pulse sensor.

 ISBN: 978-1-61208-274-5

The system went through a total of 33 states from startup to the final configuration. Analysis of the object diagrams showed that the integrity was not compromised at any time.

IV. Conclusion and Future Work

Structural changes may not be the only concern in an adaptive system when a reconfiguration is performed. In order to test state transfer, transaction handling, etc., a more rigour testing strategy is necessary. In this paper, we presented an approach to test the reconfiguration procedure of adaptive systems with respect to structural changes having regard to the special properties, such as state transfer. The iterative process model can be tailored to fit the system under test. Furthermore, the process model is designed to be an additional test activity, not a replacement for other processes and therefore focuses on the reconfiguration only, i.e., component test, etc. remain unaffected.

There are extensions to this work, which have not been discussed yet. Quality requirements, such as maximum tolerable reconfiguration time, have not been included to the test process model, which is to be addressed in future work.

References

[1] K. Nehring and P. Liggesmeyer, "Tracing structural changes of adaptive systems," in *ADAPTIVE 2010: The Second International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2010, pp. 142–145.

[2] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, SEAMS '06, 2006, pp. 2–8.

[3] H. J. Goldsby, B. H. Cheng, and J. Zhang, "AMOEBA-RT: Run-Time Verification of Adaptive Software," 2008, pp. 212–224.

[4] Component+ Partners, "Built-in testing for component-based development," in *EC IST 5th Framework Project IST-1999-20162 Component+*, Technical Report D3, 2001.

[5] F. Munoz and B. Baudry, "Artificial table testing dynamically adaptive systems," *CoRR*, abs/0903.0914, 2009.

[6] J. Zhang and B. H. C. Cheng, "Using temporal logic to specify adaptive program semantics," in *Journal of Systems and Software*, Volume 79(10), 2006, pp. 1361–1369.

[7] J. Zhang, H. J. Goldsby, and B. H. Cheng, "Modular verification of dynamically adaptive systems," in *AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development*, 2009, pp. 161–172.

 ISBN: 978-1-61208-274-5

Adaptive System Framework

A Way to a Simple Development of Adaptive Hypermedia Systems

Balík Martin and Jelínek Ivan
Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Prague, Czech Republic
e-mail: {balikm1, jelinek}@fel.cvut.cz

***Abstract*—Adaptive hypermedia systems (AHS) are complex systems that require an expensive and time-consuming design and development process. Complex solutions are usually realized as reusable frameworks and program libraries. However, there is currently no widely acceptable solution for building AHS. Based on our research, we are developing a framework that could significantly make the design and development of AHS easier. First, we formalized the adaptive system architecture, and then, we defined basic structures for storing required data. Further, we designed the adaptation and integration modules and developed reusable adaptive web user interface components. Such a framework is considered to become a foundation stone for various types of AHS.**

***Keywords*—adaptive hypermedia; personalisation; framework; software development**

I. INTRODUCTION

The purpose of adaptive hypermedia systems (AHS) is to adapt content, presentation and navigation of hypermedia to satisfy user's needs and preferences. The fundamental principle of AHS is to observe user's behavior, to build a user model reflecting all user's characteristics, e.g., knowledge, preferences, or event history, and to customize the pages presented to the user based on these characteristics. The aim of our research is to provide a reference model of AHS and its implementation that would contribute to the facilitation of an AHS development process.

The Generic Ontological Model for Adaptive Web Environments (GOMAWE) [1] forms a theoretical basis for an application framework that should rapidly simplify and speed up the AHS development. This is achieved by reusable ready-to-use software components provided by the framework implementation and extensibility of the framework for further use cases and novel technologies.

The Adaptive System Framework (ASF) was built to help a software developer create adaptive web applications. ASF provides the most typical AHS components serving as building blocks for further development. ASF is based on the theoretical model and satisfies the following important requirements. To be generally applicable, the framework has to be split into components with independent responsibilities. To follow generally accepted solutions to common application problems, design patterns [2] should be extensively used. The implementation of the framework should be based on well-known and widely used application frameworks. In contrast to other frameworks focusing on the users' collaboration [3] or adaptation process modeling [4], our framework aims at formalization of adaptive system architecture. Further, it focuses on targeting the problem of a storage layer abstraction, foundations of the data structures needed for a user modeling and providing a basic set of adaptation-oriented user interface components.

The paper is structured as follows. Section 1 deals with the description of AHS and the tasks to solve. In Section 2, a current state of the art of the discussed topic is being reviewed. In Section 3, an ASF framework is described in detail focusing on individual layers. In Section 4, both an evaluation method and application of the framework in a prototype implementation is discussed. Finally, the paper concludes by summarizing results of the research and indicates the directions of the future work; see Section 5.

II. RELATED RESEARCH

Hypermedia adaptation has become a topic of a researchers' interest for almost 15 years. The researchers have been trying to formalize an adaptive system architecture, behavior and to line up with the existing web development standards. Since self-adaptive systems are complex, they require a special approach in the process of designing and developing such projects.

In the early stage of the AHS research, the model AHAM with its implementation AHA! [5] was the first widely used architecture of the adaptive systems. The AHAM domain model can be represented by single ontology, since it deals with the concepts and their mutual relations. However, it was not designed to deal with multiple ontologies. The AHA! system is built on the basis of outdated technologies, and the web user interface is not in compliance with modern web standards. Therefore, we offer the improvements which lie in building a novel framework based on up-to-date web technologies.

In the following text, we present a layout of already existing adaptation frameworks and libraries.

GRAPPLE (Generic Responsive Adaptive Personalized Learning Environment) [6] has been developed at the Eindhoven University of Technology, as a part of the FP7 project [7]. This system is focused on adaptive learning. It is integrated with existing learning systems (e.g., Blackboard [8] or Moodle [9]). The most important contribution of this project lies in integrating the adaptive delivery of the teaching materials for the course into a supported learning process.

 ISBN: 978-1-61208-274-5

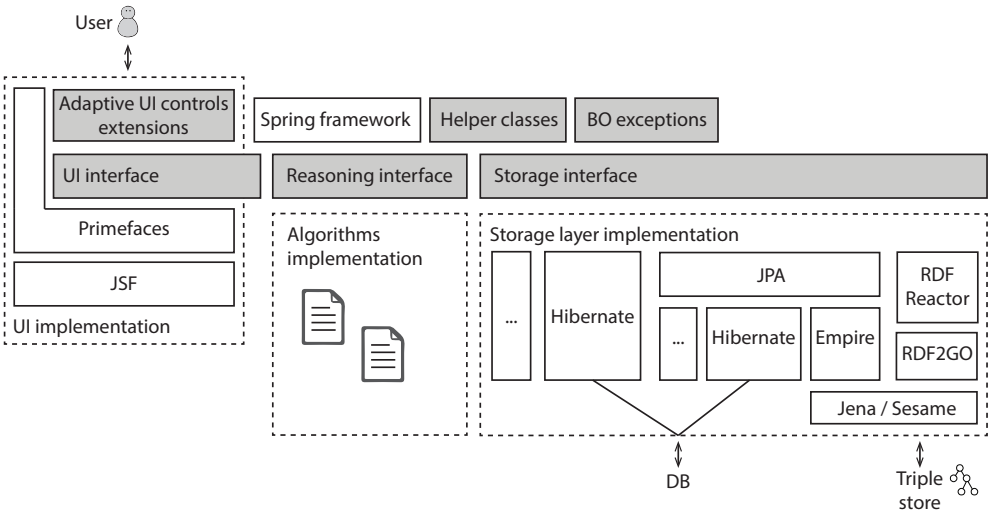


Figure 1. Adaptive system framework architecture

Another project, specifically focused on adaptive learning, is the Adaptive eLearning Platform [10]. This system is the implementation of the Virtual Apparatus Framework, a content development paradigm modeled after the process of developing a teaching lab activity. The approach used in this project is tightly connected to the learning process and is based more on pedagogical principles than on software engineering.

A further promising project is also HyperAdapt [11]. In this project, a specialized approach utilizing an aspect-oriented programming is used. The authors place the adaptivity into separate modules called adaptation aspects. The aspects are not applied on a model level, but on XML documents.

One of the solutions intended to extend legacy web applications with adaptive behavior is the Adaptive Server Framework [12]. Compared to our solution, this project is focused on server-side components only. The design principle is to separate the implementation of adaptive behavior from the server application business logic. The coupling of components is ensured by a message-based communication.

A similar solution is the Rainbow project [13]. This project uses an architecture-based approach. The system adaptation is predefined by the architecture style of the system. The commonly used design principle is the principle of a modular architecture.

Another solution partially inspiring our design of GOMAWE is the MUSE semantic framework [14]. The framework is built on multidimensional ontological planes. The intersection between the planes allows the representation of semantic rules. A similar principle is used in GOMAWE, where a multidimensional matrix of rules is used to infer the information not explicitly stored in the user model.

In comparison with other solutions, our ASF project aims at supporting not only adaptive learning, but also adaptive hypermedia systems in general. The purpose of our project is to provide a reusable solution that could be used by the developers of adaptive applications. We formally described adaptive hypermedia within the GOMAWE model and designed the framework that will be described in detail in the following sections. The Adaptive System Framework is a solution for a simpler and rapid development of AHS.

III. Framework architecture

The ASF architecture is generic. It defines an interface of various layers of the potential system, and therefore its implementation can be realized in multiple programming languages. Fig. 1 represents the description of ASF architecture based on our GOMAWE model. The ASF architecture consists of the layers replacing the GOMAWE Storage, Reasoning and UI interface layers. The main highlighted components of the core framework library define the fundamentals of the architecture. The default implementation is built on the selected persistence frameworks supporting relational database and triple stores. In our experiments the storage layer was based on various frameworks, e.g., JPA, Hibernate, Empire, or RDF Reactor. However, our framework can be extended by any other implementation of the storage interface. The extensions are indicated by the '...' symbol in the diagram; see Fig. 1. The same situation is in case of algorithms performing the adaptation above the storage layer. Some algorithms are part of the framework, others can be added by the developer as an implementation of the reasoning interface.

The user interface is based on Java Server Faces (JSF) and is supported by a Primefaces components suite [15]. Our goal is to extend basic web components by the adaptation-specific extensions; see Fig. 1. We think that there are several adaptation techniques that the framework can provide "out of the box" allowing the developers to apply the adaptation without any need of additional work.

A. Data storage

One of the most important parts of every adaptive system is the user model. This is the repository, where information about the user is stored. This information is used in the adaptation process to filter information and personalize the presentation according to the user's preferences.

 ISBN: 978-1-61208-274-5

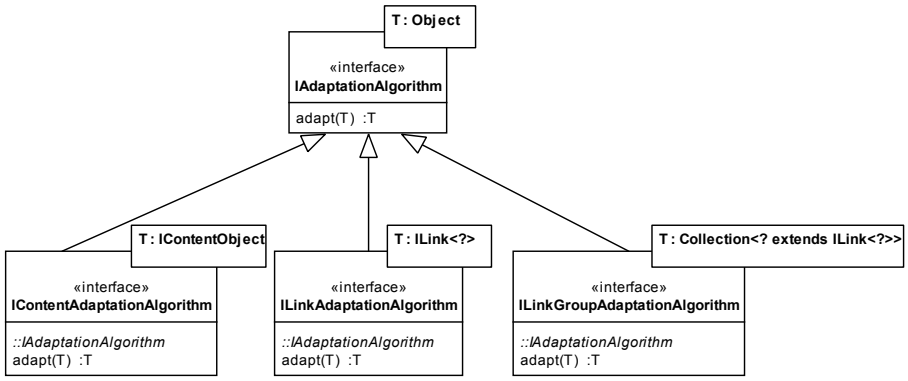


Figure 2. Adaptation algorithms classification

We divided the user data into two parts – the user profile and user model. The user profile contains explicit user's preferences. This data is corresponding to the "settings page" and is stored as key-value pairs. The key is usually a constant string defined by the developer of the application. The user model, on the other hand, stores the data observed while the application monitors the user. The information is always associated with a domain object and represents the user's relation to the object, e.g, user's knowledge of the topic, user's preferences, or their past experience. The user model corresponds to the overlay over the application domain model.

The access to the user model and user's profile is supported by an adaptation manager. The adaptation manager implementation is based both on the Singleton Design Pattern providing a manager instance and the Factory Method Design Pattern used for creating domain-specific model instances for an individual user.

In our design, the user model has a special architecture. Each attribute is assigned to one dimension. Dimensions can be custom-defined for each adaptive system. The dimension forms a group of related attributes. It can be visualized as a multidimensional matrix.

The multidimensional user model is formally described as follows:

Definition 1 (Multidimensional User Model)**.** The Multidimensional User Model is a tuple $MUM = (D, A, V)$

$$r : A \to V | \forall a \in A : r_{(a)} \in V_a \wedge r_{(a)} \in D, \qquad (1)$$

where D is a finite set of dimensions, A is a finite set of attributes, each associated with a particular dimension, V is a set of attribute values and V_a is the domain of attribute a.

Another important part of the adaptive system storage is the rule repository. The rules can represent conditions defined by the author of the content of the application, e.g., by a teacher who prepares an adaptive course, or they can be generated by specialized adaptation algorithms. The rules assume that the user-model characteristics are associated with predefined dimensions. The dimensions can be used to filter the rules while the rules are being evaluated. This contributes to better performance and helps the designer of an adaptive algorithm to maintain the rules easier.

Both the domain and the user model can be represented by simple concepts and their relations. However, our solution was designed to use multiple lightweight ontologies. The use of ontologies was motivated by the requirements of the data semantics, data exchange and integration among applications, and as well, by the need to infer the information implicitly stored in the user model.

The adaptive process is executed above the storage layer and acts as a mediator between the raw data and the user.

B. Adaptive behavior

One of the goals of the framework optimization is to make components of an adaptive system reusable and generally applicable. To achieve this requirement, we defined a general interface over any algorithm that will be used to perform the adaptation (Fig. 2).

The adaptation algorithms are further divided according to the "adaptation techniques taxonomy" (Fig. 3). A simplified version of the taxonomy is based on the taxonomy defined in [16]. Less important techniques currently not implemented by the framework were excluded from the original taxonomy. A content adaptation, link adaptation and link-group adaptation algorithms are specified in the framework.

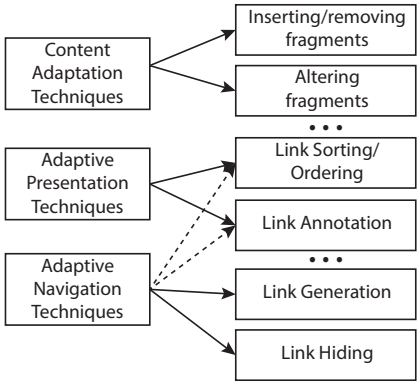


Figure 3. Simplified adaptation techniques taxonomy

A content adaptation algorithm is an algorithm which is used to transparently transform the content of the domain concepts based on the user model. It can be used to substitute the elements of the domain concepts. For example, we can recognize various extents based on the user's knowledge stereotypes (beginner, advanced, expert).

A link adaptation algorithm is intended for customizing a single link. On the other hand, the link-group adaptation algorithm assumes a collection of the links to serve as both input and result. The links can be adapted by sorting. The input collection is processed, and the order of the links is modified. A different approach is used for a link generation. In this case, the result is not dependent on the adaptation function input, since the data is retrieved from a repository. Other link adaptation strategies include a direct guidance, link annotation or adaptive link hiding.

There are two main types of the link adaptation algorithms in the framework:

- adaptation based on the current context, where the existing links can be sorted, filtered, etc.
- link generation, where the algorithm is responsible only for retrieving the input data. A default value can be provided to the adaptation function. It can be used in case when the user disables the adaptation, or if there is not sufficient input data to generate the links automatically.

In GOMAWE, the adaptation was designed as an extendable set of black box components that perform the adaptation based on the information stored in the user model. A framework implementation (Fig. 4) is realized as a Strategy Design Pattern [2]. The intent of the Strategy Design Pattern is, first, to define a family of algorithms, second, encapsulate each of them, and third, make them interchangeable. It enables the algorithm to vary independently of the clients that use it.

We can define an elementary adaptation as an adaptation function.

Definition 2 (Adaptation Function)**.** An Adaptation Function AF is a transformation between default and adapted hypermedia elements. A hypermedia element is considered as a portion of HTML code that is a part of the web page.

$$AF : e_d \rightarrow e_a, \tag{2}$$

where e_d is the default element, and e_a is the adapted element.

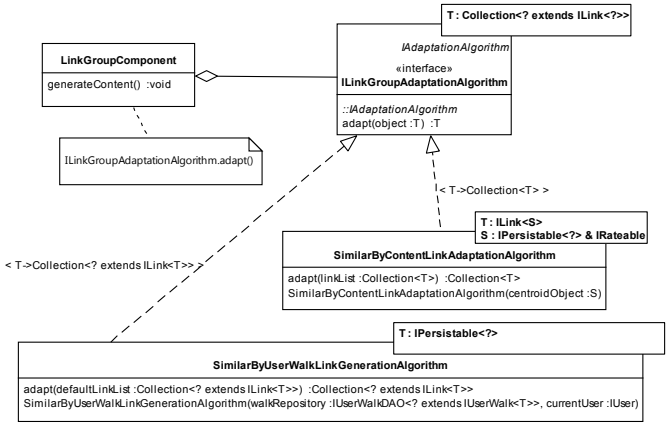


Figure 4. Link-group adaptation algorithms

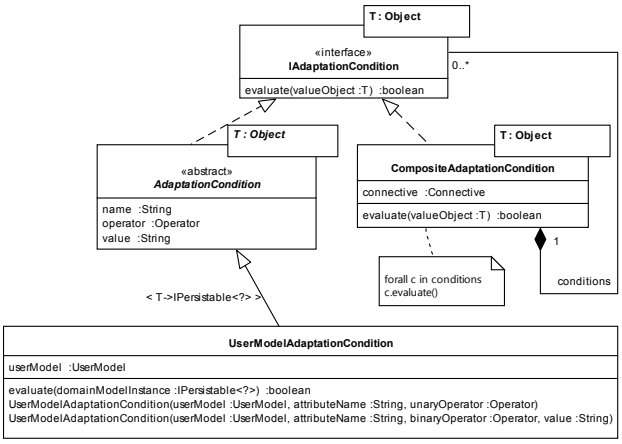


Figure 5. Condition class hierarchy

A generic group adaptation function usually takes a collection of default or initial values as an input, and returns an adapted collection of items of the same type. The particular algorithm is encapsulated inside the black box which is implemented as a class. Algorithm instances can be optionally parametrized before the actual adaptation is performed.

Another extension of the adaptation component will lie in the meta-adaptation support, where best-suited algorithms will be adaptively selected. For this purpose, the Strategy Design Pattern will be extended to the Adaptive Strategy Design Pattern [17]. The Adaptive Strategy Design Pattern defines a self-adaptive strategy. A single strategy referencing the best available concrete strategy is exposed to the client and the client is required only to provide an access to the environment information that can be used to choose the best strategy.

The data filtering in the user model is based on the conditions (Fig. 5). Condition classes follow the Composite Design Pattern [2]. The task of the Composite Design Pattern is to compose objects into tree-like structures to represent part-whole hierarchies. The Composite allows clients to treat individual objects and the compositions of objects uniformly. The conditions have multiple applications in the framework. The same hierarchy is used for evaluating the rules. The purpose of the condition is determined by a particular implementation of the abstract *AdaptationCondition* class.

The rules are defined on the intersections of the user model dimensions. The dimensions are used to limit the information space and to contribute to better evaluation performance. There are two ways of creating the rules in the data storage. The rules can be defined directly by the content designer, or they can be a product of an adaptation algorithm. The combination of these techniques can lead to interesting adaptive behavior. This will be the objective of our future research.

In the following sub-section, we will show an example of an adaptation algorithm that can be integrated into the system.

 ISBN: 978-1-61208-274-5

$document \leftarrow currentWalk$.get($currentWalk$.size$-1$)
for all userWalk in walks **do**
  $newIndex \leftarrow currentWalk$.getPosition($document$)
  $oldIndex \leftarrow userWalk$.getPosition($document$)
  {compare a similarity of documents preceding the current one in the current and a stored walk}
  while $newIndex \geq 0 \wedge oldIndex \geq 0$ **do**
    if $currentWalkDoc \neq userWalkDoc$ **then**
      break
    else
      increment quality and decrement indexes
    end if
  end while
  if $quality > 0$ **then**
    put into document-quality map
  end if
end for
sort document quality map by document quality
return set of documents sorted by quality

Figure 6. Algorithm: Get possible subsequent documents by comparing the history of the user walk transitions

Example (Walking the document space)**.**
To clerify a link adaptation, we will show how the adaptive link generation based on the similarity of navigating paths within the document space is supported by the framework.

Let us have a finite set D of the documents d. A document walk is an ordered set W, where $W \subseteq P$. The document walk is always associated with the user and represents a navigation sequence of the user throughout the document set in a single session. Finding a similar sequence allows us to predict the next most suitable document for a current user based on other users' behavior (Fig. 6).

The desired algorithm is classified as a link-group adaptation algorithm based on ASF (Fig. 2). In our case, it is a link generation algorithm.

Fig. 7 presents a sequence diagram describing typical steps of the adaptation algorithm sequencing. In our specific case, the user walk algorithm, first, requests the default values from the user model (in case of a link generation, this step is not required), and, second, it loads other user walks from the repository. Based on this data, a set of recommended subsequent documents is returned to the content generator.

C. User interface

The most user-oriented layer of the framework comprises the user interface components that are customized for the web-page adaptation. The components are based on the JSF and the Primefaces component suite. The components utilize JavaScript and AJAX to provide rich user experience. An added value to the commonly used web components is the tight coupling with adaptive behavior, user model and the adaptation engine.

An adaptive text output can be taken as an example of a simple component. In a common web component framework, we can find a text output component generating a text to the web page. The text content selection or customization must be done by the developer. In our framework, we want to provide intelligent web components tightly bound with the adaptation engine. The adaptive text component is able to provide various content adaptation techniques. The functionality of the component should be based on the configuration, selected algorithm and on the provided data storage for the data binding. Any of these parameters can be changed later, without any significant modifications to the web page logic and code.

IV. PROTOTYPE VALIDATION

The Adaptive System Framework was applied in the development of a learning course. We chose an adaptive learning environment since the adaptation is very often applied in this area. The university environment provides many opportunities to evaluate such an application in the courses and seminars. Our adaptive learning application was used in a C language programming course.

The implementation of the storage layer was based on JPA API and Hibernate implementation. MySQL Server database was used as a data storage. The choice of the

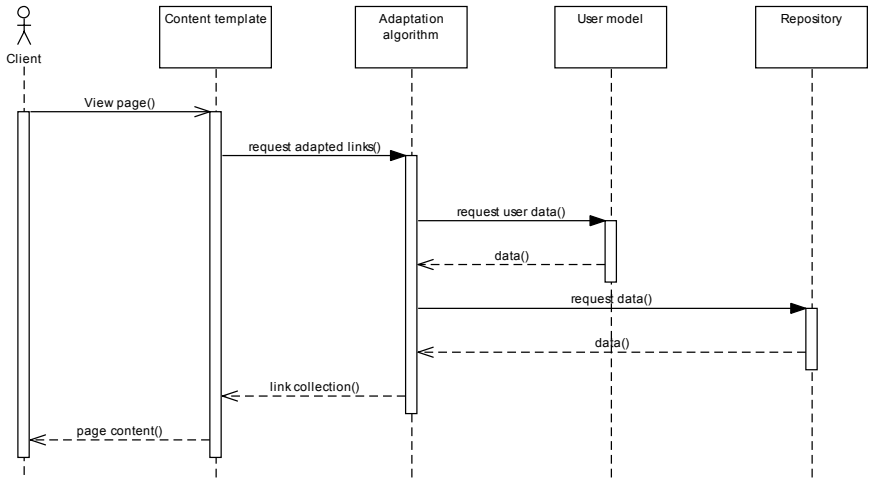


Figure 7. Adaptation algorithm usage

storage limited some benefits of the framework. However, as a prototype, it was sufficient enough to validate the framework architecture. At present, we are working on the future extensions of the adaptive application, where the ontologies representing an important feature of our design will be used. In the next version of the learning course, the data will be saved in a triple store and the integration features will be evaluated.

The centralized user model management is beneficial for the application development. Using the adaptation manager, the user profile and user model properties can be accessed from any component of the application.

The design of the application core based on the ASF framework consists of the following important steps:

1) Definition of the domain objects – in case of the learning application called learning objects and their relations
2) Definition of the user profile and user model attributes
3) Design of the adaptive algorithms for the desired behavior
4) Configuration of data sources
5) Binding the data results either to the application logic or directly to the adaptive UI components

The application was tested by 35 students, and the content was limited to one topic of one week of the semester. From the results of the log analysis, we could get interesting observations regarding the feedback from the students. In this phase, the feedback was limited to a preference screen only. While using an online course, 5 students (14%) tried to change the adaptation setting and 5 of them tried to reset the result statistics. More students were interested in personal settings, particularly, the visual theme of the application. 12 students (34%) changed the visual theme. From these results, we can conclude that a default setting of the adaptation is very important and that the adaptation based on automatic observations of the users should be extensively applied. An explicit feedback can be expected after the users become more friendly with the system and start customizing the system to be more comfortable to use.

V. Conclusions and future work

The Adaptive System Framework can be regarded as a possible solution to an effective development of adaptive hypermedia systems. The proposed framework defines fundamental components and is based on a formal model. It provides various possibilities of its implementations and their further extensions. It leads to a simplified process of the development and, at the same time, it does not limit the developer in customized extensions. The default framework implementation is based on both modern frameworks used for the development of the web applications and state-of-the-art technologies of the Semantic Web.

We have verified the framework by implementing a prototype of e-learning web application. In the future, we would like to extend the framework with more reusable adaptable JSF components based on the well-known adaptation techniques. Extending a set of implemented algorithms will enable use new methods of the personalization. The results will be thoroughly verified by the application of an adaptive web-based learning in real-class scenarios.

Acknowledgment

The results of our research form a part of the scientific work of a special research group WEBING [18]. The work was supported by the grant of the Grant Agency of the Czech Technical University in Prague.

References

[1] M. Balík and I. Jelínek, "Towards Semantic Web-based Adaptive Hypermedia Model," in ESWC Ph.D. Symposium, Tenerife, Spain, 2008, pp. 1–5.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] M. Šimko, M. Barla, and M. Bieliková, "ALEF : A Framework for Adaptive Web-Based Learning 2.0," Ifip International Federation For Information Processing, vol. 324, 2010, pp. 367–378.

[4] E. Knutov, P. De Bra, and M. Pechenizkiy, "Generic Adaptation Framework: A Process-Oriented Perspective," Journal Of Digital Information, vol. 12, no. 1, 2011.

[5] H. Wu, E. de Kort, and P. De Bra, "Design issues for general-purpose adaptive hypermedia systems," in Proceedings of the twelfth ACM conference on Hypertext and Hypermedia - HYPERTEXT '01. New York, New York, USA: ACM Press, 2001, pp. 141–150.

[6] P. De Bra, D. Smits, K. V. D. Sluijs, A. I. Cristea, and M. Hendrix, "GRAPPLE: Personalization and adaptation in learning management systems," in Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA 2010), Toronto, Canada, 2010, pp. 3029–3038.

[7] GRAPPLE Project Website. [Accessed: Apr. 24, 2013]. [Online]. Available: http://www.grapple-project.org/

[8] Blackboard Learning System Website. [Accessed: May 6, 2013]. [Online]. Available: http://www.blackboard.com/

[9] Moodle Learning System Website. [Accessed: May 6, 2013]. [Online]. Available: https://moodle.org/

[10] D. Ben-Naim, "A Software Architecture that Promotes Pedagogical Ownership in Intelligent Tutoring Systems," Ph.D. dissertation, University of New South Wales, Sydney, Australia, 2010.

[11] M. Niederhausen, S. Karol, U. Aß mann, and K. Meiß ner, "HyperAdapt: Enabling Aspects for XML," in Web Engineering, 9th International Conference, ICWE 2009, ser. Lecture Notes in Computer Science, M. Gaedke, M. Grossniklaus, and O. Díaz, Eds. San Sebastián: Springer, 2009, pp. 461–464.

[12] I. Gorton, Y. Liu, and N. Trivedi, "An extensible and lightweight architecture for adaptive server," Softw., Pract. Exper., vol. 38, no. 8, 2008, pp. 853–883.

[13] S. Cheng, "Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, 2008.

[14] F. Carmagnola, F. Cena, C. Gena, and I. Torre, "MUSE: A Multidimensional Semantic Environment for Adaptive Hypermedia Systems," in Proceedings of Lernen, Wissensentdeckung und Adaptivität (LWA), M. Bauer, B. Brandherm, J. Fürnkranz, G. Grieser, A. Hotho, A. Jedlitschka, and A. Kröner, Eds. Saarbrücken, Germany: DFKI, 2005, pp. 14–19.

[15] PrimeFaces Component Suite. [Accessed: Jan. 31, 2013]. [Online]. Available: http://primefaces.org

[16] E. Knutov, P. De Bra, and M. Pechenizkiy, "AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques," New Review of Hypermedia and Multimedia, vol. 15, no. 1, Apr. 2009, pp. 5–38.

[17] O. Aubert and A. Beugnard, "Adaptive Strategy Design Pattern," in Proceedings of The Second Asian Pacific Pattern Languages of Programming Conference (KoalaPLoP 2001), The Country Place, Melbourne, 2001, pp. 1–12.

[18] Webing Research Group Website. [Accessed: May 6, 2013]. [Online]. Available: http://webing.felk.cvut.cz

 ISBN: 978-1-61208-274-5

A FPGA Implementation of Prediction Error Method for Adaptive Feedback Cancellation using Xilinx System Generator™

Marius Rotaru, Cristian Stanciu, Silviu Ciochină
Dept. of Telecommunications
University Politehnica of Bucharest
Bucharest, Romania
marius.rotaru@gmail.com, {cristian, silviu}@comm.pub.ro

Felix Albu, Henri Coandă
Electrical Engineering Department
Valahia University of Targoviste
Targoviste, Romania
{felix.albu, coanda}@valahia.ro

***Abstract*—This paper describes a real-time, field programmable gate array (FPGA) implementation of Feedback Cancellation (FC) system to improve the intra-cabin communication among the driver and passengers, which is typically degraded by the noisy environment and by the distance in between them. The feedback canceller, used to reduce the acoustic coupling the loudspeaker and the microphone, is based on the continuously adaptive filtering technique, implementing the prediction error method (PEM) for closed loop system identification. The adaptive algorithm implements the modified least mean square algorithm (MLMS) while for the linear prediction a fix-order linear predictor has been selected. The implementation was done using Xilinx System Generator™ (XSG)**

Keywords-Adaptive Algorithm; Adaptive Feedback Cancellation; Prediction Error; MLMS; FPGA; Xilinx System Generator

I. INTRODUCTION

The acoustic feedback is a major problem of audio processing field, occurring whenever the sound is captured and reproduced in the same environment. The communication in vans and limousines between the passengers in the front and the rear is degraded due to the presence of the noise as well as the long distance between them [1]. This can be improved by using a speech reinforcement system. The simplest, one channel speech reinforcement system, picks-up the speech using a microphone, amplifies it and then plays it back to a loudspeaker. Due to the electro-acoustic coupling between loudspeakers and microphone, a closed-loop system is created. To avoid the instability (howling) of the system, a feedback canceller has to be used.

Different methods attempting to minimize the effect of acoustic feedback have been proposed in literature. They are broadly classified as feedforward suppression and feedback cancelation techniques. For the case of feedforward suppression technique, the use of notch-filter based howling suppression (NHS) represents a traditional and robust solution [1-3]. The main disadvantage of this method is that it is reactive: in order to identify and eliminate the oscillation frequencies the howling must firstly occur. A more promising solution is to use the adaptive feedback cancellation (AFC) method, which is based on the estimation

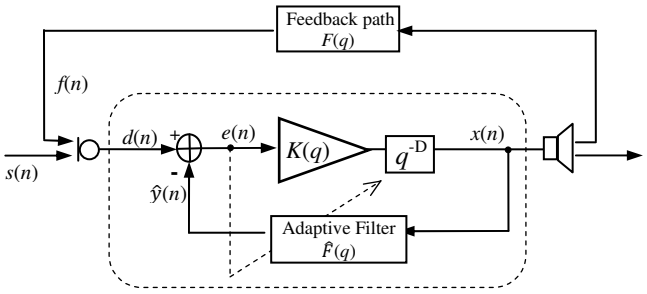


Figure 1. Adaptive Feedback cancellation structure

of acoustic feedback path, belongings to the class of room modeling methods. As illustrated in the Fig. 1, the feedback canceller $\hat{F}(q)$ produces an estimate $\hat{y}(n)$ of the feedback signal $f(n)$, obtained by filtering the loudspeaker signal $x(n)$ with $F(q)$, and subtracts it from the microphone signal $d(n)$ so that ideally the clean speech signal $s(n)$ amplified by a factor K and played back to the loudspeaker. Depending on the quality of the feedback path estimation, the feedback is almost eliminated. The most robust method to eliminate it is based on the closed loop identification theory [4] - [5]. The direct method for closed loop identification [5], named prediction-error-method (PEM) is a promising proactive solution for AFC. Prediction error for AFC (PEMAFC) algorithms has been deeply analyzed in context hearing aids applications [6-10]. Also, recently the PEMAFC method has been tested in the car scenario case [11-12]. In this paper we propose a FPGA implementation of the PEMAFC algorithm in Xilinx System Generator™, based on the configuration from [8-9] using a fix model of input signal.

The paper is organized as follows: Section II is dedicated to the description of the PEMAFC algorithm implemented on FPGA. The Section III covers the FPGA implementation aspects. In Section IV the experiments as well as the results are reported. Section V presents the conclusions of the work.

II. PREDICTION ERROR METHOD FOR AFC SYSTEM

The notation from [8] has been adopted: q^{-1} denotes the unitary delay, so that $q^{-1}\ u(n) = u(n-1)$. A discrete-time filter with filter length L is represented as a polynomial $F(q)$ in q, i.e.,

 ISBN: 978-1-61208-274-5

$$F(q) = f_0 + f_1 q^{-1} + \cdots + f_{L-1} q^{-L+1}, \quad (1)$$

or by its vector $\mathbf{f}=[f_0, f_1 \ldots f_{L-1}]^T$, so that the filtering operation consists of applying the polynomial to the input sequence:

$$F(q)x(n) = \mathbf{f}^T\mathbf{x}(n), \quad (2)$$

with $\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(n-L+1)]^T$.

As proposed in the [6], [7] and reiterated in [8], the direct method of the closed-loop identification of feedback path $F(q)$ and the desired signal model $H(q)$ is presented in the Fig.2. The main assumption of this method starts from the fact that the desired signal $d(n)$ can be modeled as a $H(q)w(n)$, where $w(n)$ is the white noise signal and the $H(q)$ is the desired signal model, which is inversely stable ($A(q)=H^{-1}(q)$). In such case the bias in the feedback path estimation can be eliminated by decorrelating the signal of the adaptive algorithm by passing them through $A(q)$. In case of PEM, the feedback path $F(q)$ and the desired signal model $H(q)$ are estimated by minimizing the energy of so calling prediction error $e_p(n)$:

$$e_p(n) = \hat{H}^{-1}(q)(d(n) - \hat{F}(q)x(n)), \quad (3)$$

i.e., $$J(\hat{\mathbf{f}}(n)) = E\{|\hat{H}^{-1}(q)(d(n) - \hat{F}(q)x(n))|^2\}$$
$$= E\{|(d_p(n) - \hat{\mathbf{f}}^T(n)\mathbf{x}_p(n))|^2\}. \quad (4)$$

Minimizing (4) generates:

$$\hat{\mathbf{f}}(n) = E\{\mathbf{x}_p(n)\mathbf{x}_p^T(n)\}^{-1} E\{\mathbf{x}_p(n)d_p(n)\}. \quad (5)$$

Writing $d_p(n)$ as:

$$d_p(n) = \hat{H}^{-1}(q)s(n) + F(q)x(n). \quad (6)$$

When $H(q) = \hat{H}(q)$,

$$d_p(n) = w(n) + F(q)x(n), \quad (7)$$

and the speech signal $s(n)$ is converted to a white noise signal $w(n)$, resulting an unbiased feedback path estimate.

From (4) it can be observed that minimizing $J(\hat{\mathbf{f}}(n))$ is equivalent with performing an adaptive filtering on the decorrelated (pre-whitened) signals $\{d_p(n), x_p(n)\}$ or equivalently on $\{e_p(n), x_p(n)\}$.

Both fixed and adaptive estimates of the desired signal model $H^{-1}(q)$ have been considered in literature [9]. Since our attention is focused on the FPGA implementation, in this paper we choose the fixed model, representing the averaging of speech spectrum, defined by:

$$H(q) = \frac{1}{1-\alpha q^{-1}}, \quad |\alpha| < 1 \quad (8)$$

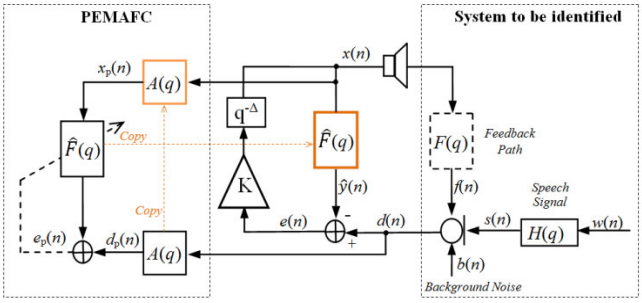


Figure 2. Feedback cancellation with prediction error method

The adaptive filtering algorithm selected to perform these minimization is the Modified LMS (MLMS) [13], which is particularly suited for adaptive systems whose performance suffers from the presence of strong target signals (such as speech) that exhibit large fluctuations in short-time power levels. The sum version of this algorithm has been selected, with the following formula for updating the filter coefficients:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + e(n)\mathbf{x}(n)f[x(n), e(n)], \quad (8)$$

$$f[x(n), e(n)] = \mu(n) = \frac{\bar{\mu}}{L(\hat{\sigma}_e^2(n) + \hat{\sigma}_x^2(n)) + \varepsilon}, \quad (9)$$

where L is the length of adaptive filter, $\bar{\mu}$ is an adaptation constant and $\hat{\sigma}_e^2$ and $\hat{\sigma}_x^2$ are the power estimates of the error respectively of the reference signal. They can be obtained as follow:

$$\hat{\sigma}_{e,x}^2(n) = \lambda\hat{\sigma}_{e,x}^2(n-1) + (1-\lambda)(e^2(n), x^2(n)), \quad (10)$$

where λ is a weighting factor chosen as λ =1-1/(KL), with K>1. A Summary of PEMAFC algorithm is described in the Table.1.

TABLE I. TIME-DOMAIN PEMAFC ALGORITHM FOR THE FIXED ORDER LP

Initialization:

L, ε, $\bar{\mu}$, λ, $\hat{\mathbf{f}}(\mathbf{0})=\mathbf{0}$, $e(0)=0$,

Computation: for each input sample $x(n)$, n= 1, 2,. . .

$e(n) = d(n) - \hat{\mathbf{f}}^T(n)\mathbf{x}(n)$;

$x(n) = K \cdot e(n-D)$

Pre-filter $e(n)$ and $x(n)$ with $A(q)$

$$e_p(n) = e(n) - \alpha e(n-1), \quad x_p(n) = x(n) - \alpha x(n-1),$$

Estimate the power of pre-filtered signals

$$\hat{\sigma}^2(n) = \lambda\hat{\sigma}^2(n-1) + (1-\lambda)(e_p^2(n) + x_p^2(n)),$$

Update the coefficients

$$\hat{\mathbf{f}}(n+1) = \hat{\mathbf{f}}(n) + \frac{\bar{\mu}}{L\hat{\sigma}^2 + \varepsilon} e_p(n)\mathbf{x}_p(n).$$

End

III. FPGA IMPLEMENTATION

This section describes the adaptation of the PEMAFC algorithm on a realizable hardware platform suitable for automotive environment. Considering the PEMAFC based Feedback Cancellers a subcomponent of a speech reinforcement system including a noise cancellation component as well as a voice activity detector, the target platform must be cost effective with a relatively high performance DSP. The Xilinx Automotive (XA) Spartan®-6 families of FPGA was used [14].

A. Design Process

The PEMAFC algorithm was originally developed as MATLAB scripts using high precision floating-point arithmetic. The SoNoScout NVH Binaural recording and analysis system 653A and Sound Level Meter 2250L were also used. A one-to-one conversion to an equivalent FPGA implementation cannot be directly or easily implemented with reasonable resource utilization. Also, the precision of data in the FPGA implementation is limited to a fixed number of bits (fixed-point representation) which results in the addition of quantization errors to the system.

The first step of the implementation consists of identifying the parts of the algorithm, which became the main blocks of the hardware. A fixed point version of these blocks has been implemented using Fixed-Point Toolbox™ in Matlab® [15] and then compared to the floating point version.

Following the Matlab fixed point validation, a Xilinx System Generator™ (XSG) [16] model was developed. Each block has been individually validated by passing the data to/from Matlab workspace as well as using the "scope" block in the model by connecting the signal of interest to it. Once the XSG model has been created it has been validated against Matlab implementation. Finally, the design has been synthesized using the Xilinx ISE 13.4 design suite and run on the FPGA target in the "hardware-in-the-loop co-simulation".

B. The Hardware implementation

The implementation of the PEMAFC algorithm is done on a Spartan6 FPGA, i.e., the XC6SLX45[15]. The system clock frequency is approximately 100 MHz and the sampling frequency is 16 kHz; consequently, there are approximately 6250 clock periods available between two successive samples.

The flow of the algorithm is described in Fig.3. In the first phase, the output of the adaptive MAC filter $y(n)$ is generated and the error signal $e(n)$ is computed. In the second phase a fixed order pre-whitening of error signal $e(n)$ and its delayed and amplified version $x(n)$ is realized based on (8). A power estimate (10) of both decorrelated error and reference signals is generated in the third step. In the fourth step the algorithm's step size is computed based on the previously power estimated values (9). The last phase is dedicated updating the adaptive filter's coefficients based on their past values, decorrelated reference and error signals $\mathbf{x}_p(n)$ respectively $e_p(n)$ and the previously computed step size. The FPGA implementation of the PEMAFC algorithm requires a few RAM memory blocks, as follows.

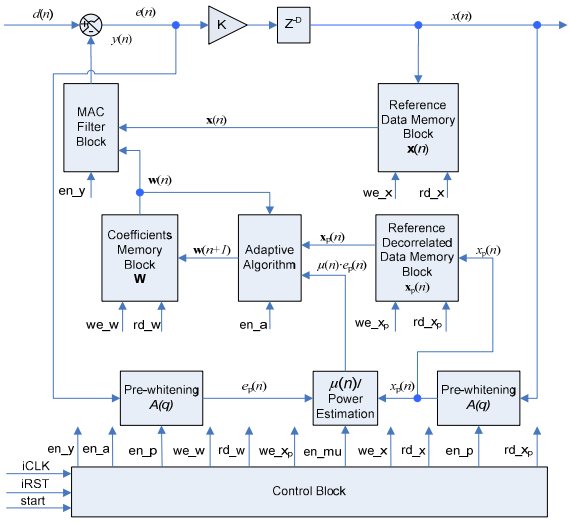


Figure 3. Implementation scheme of the PEMAFC

The first one is associated with the reference signal samples; this memory can be viewed as $L\times1$ matrix. The second one is associated with the decorrelated reference signal samples with the same depth as the previous one. The third memory block is used to keep the filter coefficients.

Only one division is associated with the PEMAFC algorithm (9). The implementation was based on the Xilinx Divider Generator 3.0 block (radix 2 non-restoring division version) instead of the CORDIC Divider block, which is much more resource consumer. For a better precision, the division has been done between $e_p(n)$ and the power estimation $\hat{\sigma}$ (both operands being reinterpreted as signed integers), the result being scaled by $\log_2(L)$.

By selecting $\bar{\mu}$ as power of two, only four multipliers cells are used to implement the PEMAFC algorithm. Two of them are used in a pipelined manner, i.e., series of L computations (one for updating the filter coefficients and the other for computing the output of the FIR filter in a multiply with accumulate mode). The other two are involved in the power estimation of $e_p(n)$ and $x_p(n)$ signals. The input and the output signals $d(n)$ and $x(n)$ are represented on 16 bits (Q15 representation) while the internal representation at different stages varies based on the dynamic range of internal "signals", in order to avoid the overflows and to minimize the quantization errors.

IV. RESOURCE USAGE AND SIMULATION RESULTS

The functional simulations have been done using FPGA target in the hardware-in-the-loop co-simulation mode as in the Fig.4. The real signal was read from Matlab workspace and it was either a voice signal with an additive white Gaussian noise (SNR=30dB) or an auto-regressive noise generated by passing a white Gaussian noise through a 10 order AR system.

The sampling frequency was 16kHz and the adaptive algorithm parameters used are: $L = 256$ (the same length as feedback path); fix step size $\bar{\mu} = 2^{-6}$; regularization factor

 ISBN: 978-1-61208-274-5

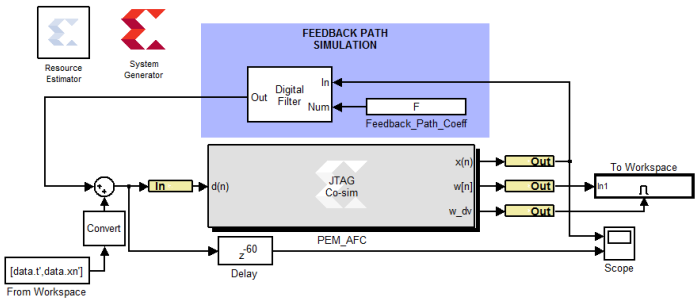


Figure 4. Hardware-in-the-loop co-simulation of the PEMAFC

ε=10^{-3}; weighting factor for power estimation $\lambda = 0.9961$; the parameter of fixed model $\alpha = 0.91$.

The gain and the delay of the forward path are K=10dB and D=60 samples (3.75ms) respectively. The feedback path was simulated using Simulink FIR Filter block.

The performance measure was the normalized misalignment (in dB), defined as $20\log_{10}\|\mathbf{f} - \hat{\mathbf{f}}(n)\|_2 / \|\mathbf{f}\|_2$, where $\mathbf{f}$ is the true impulse response of the feedback path and $\|\bullet\|_2$ denotes the l2 norm. The effect of the quantization error on the AFC performance is shown in Fig. 5.

Table II shows the resource requirement of the FPGA implementation as reported by the Xilinx ISE Foundation.

V. CONCLUSIONS

In this paper, a sequential time based, PEMAFC algorithm is implemented on FPGA using Xilinx System Generator. Comparable results with infinite precision version have been obtained. Resource analysis shows the design uses only 15% of the total available general logic resources making possible an integration with other in-car sub-systems on a single FPGA. The implementation of adaptive estimate of signal model will be considered in the future work.

ACKNOWLEDGMENT

This work was supported under the Grant POSDRU/107/1.5/S/76813 and Grant CNCS-UEFISCDI PN-II-ID-PCE-2011-3-0097.

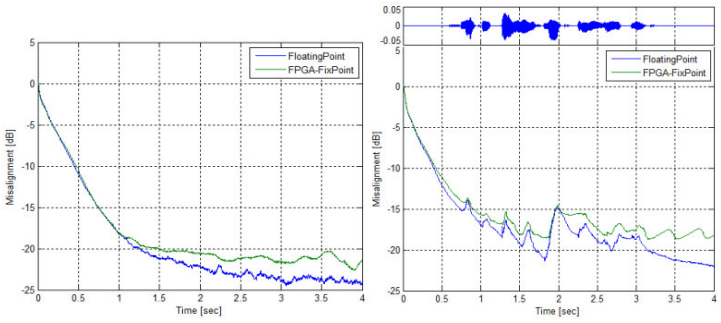


Figure 5. Misalignment of the PEMAFC (finite and infinite precision). a) auto-regressive noise; b) voice signal.

TABLE II. RESOURSE UTILIZATION FOR PEMAFC ALGORITHM WITH FIXED LP ORDER.

Available Resources (total)	Used Resources
Slices (6822)	896 (15%)
FFs (54.576)	3100 (5%)
4-LUTs (27288)	2895 (10%)
RAMB8B (323)	3 (1%)
DSP48A1s (58)	4 (6%)

REFERENCES

[1] E. Hänsler and G. Schmidt, "Acoustic Echo and Noise Control: A Practical Approach," John Wiley & Sons, New York, NY, USA, 2004.

[2] J. Chang and J.R. Glover, "The feedback adaptive line enhancer: a constrained IIR adaptive filter," IEEE Trans. Signal Process., vol. 41, Nov. 1993, pp. 3161–3166.

[3] P. Gil-Cacho, T. van Waterschoot, M. Moonen, and S. H. Jensen, "Regularized Adaptive Notch Filters for Acoustic Howling Suppression," Proceedings of 17th European Signal Process. Conf. (EUSIPCO '09), pp. 2574–2578.

[4] L. Ljung, "System Identification: Theory for the User," Prentice Hall PTR, 1998.

[5] U. Forssell, "Closed-loop Identication: Methods, Theory, and Applications", Dissertations No. 566, Linköping 1999.

[6] J. Hellgren and U. Forssell, "Bias of feedback cancellation algorithms in hearing aids based on direct closed loop identification," IEEE Trans on Speech and Audio Processing, vol. 9, Nov. 2001, pp. 906–913.

[7] J. Hellgren, "Analysis of feedback cancellation in hearing aids with filtered-X LMS and the direct method of closed loop identification," IEEETrans. Speech Audio Process., vol. 10, Feb. 2002, pp. 119–131.

[8] A.Spriet, I. Proudler, J. Wouters, and M. Moonen,"Adaptive feedback cancellation in hearing aids with linear prediction of the desired signal," IEEE Trans on Signal Processing, vol. 53, Oct. 2005, pp. 3749-3763.

[9] Ann Spriet, S. Doclo, Marc Moonen, and Jan Wouters, "Feedback Control in Hearing Aids," in Springer Handbook of Speech Processing, pp. 979–999. Springer Verlag, 2008.

[10] M. Rotaru, F. Albu, and H. Coanda, "A variable step size modified decorrelated NLMS algorithm for adaptive feedback cancellation in hearing aids," in Proc. ISETC, Timisoara, Romania, 2012, pp. 263–266.

[11] A. Ortega, E. Lleida, E. Masgrau, L.Buera, and A. Miguel "Acoustic Feedback Cancellation in Speech Reinforcement Systems for Vehicles" INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, ISCA, Sep. 2005, pp. 2061-2064.

[12] S. Cifani , L. C. Montesi, R. Rotili, E. Principi, S. Squartini, and F. Piazza, "A PEM-AFROW based algorithm for Acoustic Feedback Control in Automotive Speech Reinforcement Systems," Proceedings of 6th International Symposium on Image and Signal Processing and Analysis, Sep. 2009, pp. 656–661.

[13] J. E. Greenberg, "Modified LMS algorithms for speech processing with an adaptive noise canceller," IEEE Trans. Speech Audio Process., vol. 6, no. 4, Jul. 1998, pp. 338–350

[14] Xilinx Inc., "XA Spartan-6 Automotive FPGA Family Overview," DS170 (v1.2), Dec. 2011.

[15] MathWorks, "Fixed-Point Toolbox," http://www.mathworks.com/products/fixed, [retrieved: March, 2013].

[16] Xilinx Inc., "Xilinx System Generator for DSP," http://www.xilinx.com/tools/sysgen.htm, [retrieved: March, 2013].

A Software Infrastructure for Executing Adaptive Daily Routines in Smart Automation Environments

Estefanía Serral
Christian Doppler Laboratory
for Software Engineering Integration (CDL)
Vienna University of Technology
Email:estefania.serral@tuwien.ac.at

Pedro Valderas and Vicente Pelechano
ProS Research Center
Universitat Politècnica de València
Email:{pvalderas, pele}@pros.upv.es

***Abstract*—Since the advent of Pervasive Computing, the execution of user daily routines in an adaptive way has been a widely pursued challenge. Its achievement would not only reduce the tasks that users must perform every day, but it would also perform them in a more convenient way while optimizing natural resource consumption. In this work, we meet this challenge by providing a software infrastructure. It allows users' routines to be automated in a non-intrusive way by taking into account users' automation desires and demands. We demonstrate this by performing a case-study based evaluation.**

***Index Terms*—adaptive routine automation; models at runtime;**

I. Introduction

In recent decades, computers have become more and more common in many items such as ovens, refrigerators, coffee makers, mobile phones, tablets, etc. This proliferation of technology brings the building of smart environments closer to becoming a reality. Smart environments provide services to control the items that are used in our daily activities [1]. For instance, there are pervasive services for controlling lights, air conditioner and heating, windows, coffee makers, etc.

One of the final goals of developing smart environments is to automate user daily routines by using these services. A routine is a set of tasks characterized by habitual repetition in similar contexts. For instance, a typical routine could be the following. Every working day at 8 o'clock, Bob's alarm clock goes off. Bob wakes up, switches the lights on and stops the alarm. Then, to take a shower, Bob turns on the bathroom heating if it is cold. Finally, after getting dressed, Bob goes to the kitchen and makes a coffee for breakfast.

Due to the fact that people are creatures of habit, we perform numerous daily routines such as the one presented above. Several works, such as [2][3][4] [5], have dealt with performing these routines on the users' behalf; however, their solutions may lead to intrusive systems that automate tasks that users do not necessarily want automated. In this work, we present a software infrastructure capable of automating users routines in a non-intrusive way. Due to the complexity of human behaviour, user participation is necessary in order to avoid intrusiveness when attempting to fulfill user demands. For this reason, the infrastructure that we propose makes use of models during runtime. These models allow routines to be represented by using high-level concepts that are close to user knowledge. This helps users to understand the routines to be automated and to participate in their design. By simply interpreting the models at runtime, the infrastructure can automate the routines as described.

With this infrastructure, we could make users' lives easier and provide them with a higher quality of life: they would not have to waste their time or worry about the tasks that could be automated (e.g., Bob will never oversleep in the morning because the alarm clock is set automatically). In addition, these tasks could be performed more efficiently and in a more convenient way for users since tasks can be analyzed before being automated using the models (e.g., heating could be turned on 10 minutes early so that the bathroom is already hot when Bob takes a shower; instead of the alarm clock going off, Bob's preferred radio channel could be used). Moreover, routines could self-adapt according to context (e.g., blinds could be raised if it is a sunny day instead of switching lights on) and could help to reduce natural resource consumption by applying the advice provided by experts on controlling lighting, heating and air conditioning, taps, and so on (e.g., all lights could be automatically switched off when the inhabitants leave home; blinds could be lowered in summer when nobody is at home so that it is not so hot when the inhabitants arrive).

The rest of the paper is organized as follows. Section II describes the related work. Section III explains essential requirements for routine automation. Section IV presents the software infrastructure that automates routines in a context-adaptive way. Section V validates the approach using a case study based evaluation. Section VI concludes the paper.

II. Related Work

Related work can be subdivided into machine-learning approaches, context-aware rule-based approaches, end-user centered approaches, and task-oriented computing.

Machine-learning approaches have attempted to deal with the automation of user routines by automatically inferring them from past user actions [2][3]. These approaches have done excellent work by automatically learning from user behaviour; however, they have some drawbacks. They may be intrusive for users because they do not usually take into account users' desires (e.g., the repeated execution of an action does not imply that the user wants this automation). Also, they

 ISBN: 978-1-61208-274-5

reproduce the actions that users have frequently executed in the past and in the same manner that they were executed. This prevents user tasks from being carried out in a more efficient and convenient way and does not allow tasks to be automated if they were not performed by users (e.g., closing windows when users are not at home and it starts to rain).

Context-aware rule-based approaches have made great advances in introducing context into software systems. To automate user tasks, they program rules that trigger the sequential execution of actions when a certain context event is produced [4] [5]. However, although context information is taken into account, these works do not usually consider the personal desires of each user; therefore, they may still be annoying. Furthermore,these techniques are only appropriate for automating relatively simple tasks [6]; hence, they usually require large numbers of rules. If we also consider that these rules have to be manually programmed [6], the understanding and maintenance of the system may become very difficult.

End-user centered approaches provide alternatives for end-users to program their environments [7][8]. Most of these approaches are focused on end-user programming by presenting particular UIs and languages. These approaches generally provide better user control. However, they have limited capacities to help end-users build the automations. Therefore, they are only appropriate for developing simple tasks commonly described in the literature, such as controlling lights or doorbells.

Task-oriented computing uses task modelling to facilitate the interaction of users with the system. These systems have proven that task modelling is effective in several fields such as user interface modelling [9], assisting end-users in the execution of tasks [10], etc. These works show the growing usage of task modelling and its remarkable results and possibilities to model system behaviour. However, none of these works attempt to automate adaptive daily routines. Hence, they neither provide enough expressiveness to specify adaptive routines nor enough accuracy to allow their subsequent automation.

III. Requirements for Routine Automation

The users' tasks automation is a delicate matter. The execution of an undesired task will be intrusive for users, and may bother them, interfere in their goals, or even be dangerous; all of which would eventually cause the loss of user acceptance of the system. For instance, consider that the outside door and the security system have been programmed so that the door is automatically locked and the security system is automatically activated when the inhabitants leave home. This can be useful because they will not have to do these tasks anymore, but it can also be a burden if the inhabitants are absent-minded: they will have to unlock the outside door and deactivate the alarm every time they forget something. To prevent these intrusive situations, the following aspects are required:

- **The routines must be automated according to the users' desires and demands**. This is essential so that the routines to be automated are those that users want and are automated the way they want them to be. Due to the technical context, and the imprecise and ambiguous nature of human behaviour, it is very difficult for a system to sense or infer this information. Therefore, **the participation of the users** is necessary in order to fulfill their automation desires and demands [11].
- **The routines must be adaptive to context**. Context information is essential to be able to execute the routines in the opportune situation. For instance, in the routine used as the example, it would be intrusive if the bathroom heating is switched on when the temperature is high or if the radio is turned on anytime. Therefore, routines must be described in a context-adaptive way (e.g., the bathroom heating must be automatically switched on at 7:50 on working days only if the temperature is low, and the radio must be turned on 10 minutes later).
- **Routine adaptation must be facilitated at runtime**. Some routines might never change in user life; however, most of them will. Users' behaviour usually changes over time and the automated routines need to be adapted to these changes. Otherwise, the system may become useless and intrusive. Since these types of changes cannot be anticipated at design time, the automation of routines must be performed in such a way that their adaptation after system deployment is facilitated at runtime.

IV. The Software Infrastructure

A smart environment is developed to provide pervasive services that serve people in their everyday life. These services are in charge of interacting with physical devices in order to change the state of the environment and to sense context. On top of these services, we develop a software infrastructure (see Figure 1) that is designed to properly automate user routines satisfying the requirements explained in the previous section:

- To **facilitate user participation**, the software infrastructure makes use of design models at runtime. It provides a task model that describes the routines by using concepts of a high-level of abstraction that are close to the domain and to user knowledge (concepts such as task, preference, location, etc.). This helps end-users to participate in the routine description since it allows them to focus on the main concepts (the abstractions) without being confused by low-level details [12].
- To execute the described routines **in a context-adaptive way**, the task model describes each routine as a coordination of pervasive services that are performed in the opportune context, i.e., in a context-adaptive way. In addition, in order to be aware of the current context and to be able to automate the routines accordingly, the software infrastructure provides a context manager. It dynamically manages the context changes produced at runtime by using a context repository.
- To automate the described routines in such a way that **their adaptation after system deployment is facilitated**, the software infrastructure has an automation engine that directly interprets the task model at runtime. The model is machine-processable and precise enough to provide the infrastructure with all the information needed to execute

 ISBN: 978-1-61208-274-5

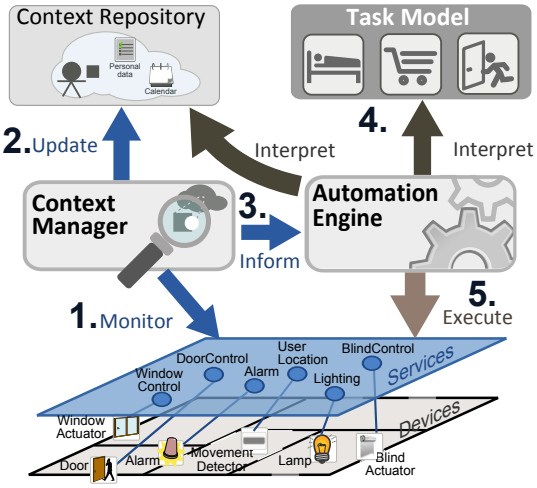


Fig. 1. Runtime infrastructure

the routines. Therefore, when a context change is detected by the context manager, it informs the engine. The engine then reads the routine information from the task model and executes the corresponding pervasive services according to context. With this strategy, the task model is the only representation of the routines to be automated. This allows the routines to be adapted by simply updating the model. As soon as it is changed to adapt the routines, the changes are also taken into account by the engine.

Thus, the infrastructure provides the following main components (see Figure 1): the *context-adaptive task model*, which describes each routine as a context-adaptive coordination of pervasive services; the *context manager*, which is in charge of monitoring context changes at runtime, updating the context repository accordingly, and informing the automation engine about the changes; and the *automation engine*, which is in charge of automating the routines in the opportune context by interpreting the models.

A. The Software Infrastructure in Execution

The software infrastructure executes the routines as described in the task model. This model allows the routines to be described precisely and at a high level of abstraction. As an example, Figure 2 shows the modelling of the routine used in the introduction (the *WakingUp* routine). The root task of the hierarchy represents the routine and is associated to a context situation, which indicates the context conditions whose fulfilment starts the execution of the routine (*WorkingDay=true AND CurrentTime=7:50*). The root task is broken down into simpler tasks (*turn on bathroom heating*, *turn on the radio*, *illuminate the room* and *make coffee*). An intermediate task must be broken down until the leaf tasks can be executed by an available pervasive service. Each leaf task must be related to a pervasive service that can carry out the task. For instance, the *turn on the radio* task is associated to a pervasive service that interacts with the radio to turn it on. This relation is established by simply indicating the service identifier.

If the tasks of the same parent are related to each other, they are carried out in a sequential order according to the indicated temporal relationships. These relationships may depend on context. Thus, in the example, the heating is turned on first; ten minutes later, the radio is turned on and the room is illuminated; and finally, a coffee is made when the user enters the kitchen.

A task can have a context precondition (represented between brackets), which defines the context conditions that must be fulfilled so that the task is performed (e.g., the *turn on bathroom heating* task is only executed if the temperature is low). If the tasks of the same parent are not related to each other, only the first task whose context precondition is satisfied is executed; e.g., to illuminate the room, if the outside brightness is low, lights will be switched on; otherwise, blinds will be raised.

To automate the routines as described in the task model, the software infrastructure performs the following steps (see Figure 3):

1) **Detecting context changes:** A context change is physically detected by a sensor, which is controlled by a pervasive service in the smart environment. The context manager monitors all these services to check context changes. For instance, the context manager monitors the Clock service to detect the time changes. When a change is detected (e.g., it is 7:50 a.m.), the manager updates the current context in the context repository and notifies the engine about this change.
2) **Checking context situations:** After receiving the notification of a context change, the engine analyzes the context situations of the routines specified in the Task Model to check if any of them depend on the context change. Then, by making use of the context manager, the engine checks if any of those context situations are

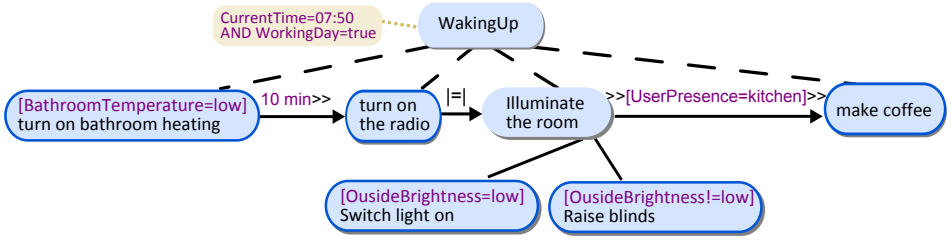


Fig. 2. Routine task for waking up the user

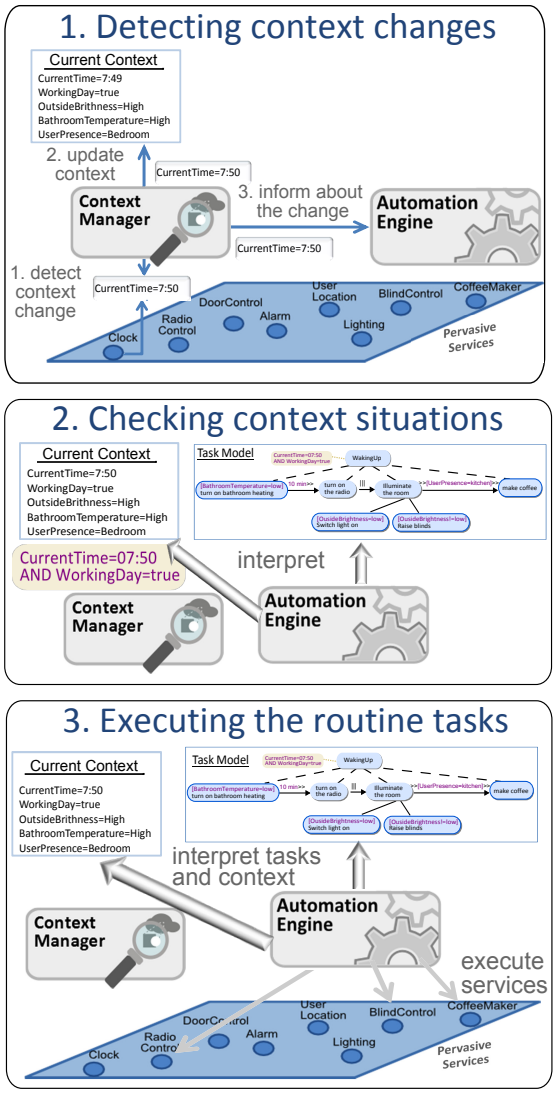


Fig. 3. A possible execution of the WakingUp routine

fulfilled. For instance, when the context manager notifies the engine that it is 7:50 a.m., the engine gets the context situations that depend on time, such as the one for the WakingUp routine, and checks them. On a working day, the engine checks that the context situation of the the WakingUp routine is satisfied.

3) **Executing the routine tasks:** The engine executes the routines whose context situation is satisfied. The engine uses the context manager to check the context conditions. To execute each routine, the engine executes its leaf tasks according to their refinements, their context conditions in the current context, and their temporal relationships. For instance, to automate the WakingUp routine, the engine gets the first subtask (*turn on bathroom heating*) and checks its precondition (*BathroomTemperature=low*). If it is true, the engine executes its related service. The engine then waits 10 minutes, as its relationship with the next task indicates. After that, the engine executes the service related to the *turn on the radio* task. The engine then gets the next task, which is the *illuminate the room* task. To execute it, the engine gets its first subtask (*switch light off*) and checks its context precondition (*OutsideBrightness=low*). If it is satisfied, the engine executes the service related to the task for switching lights on. Otherwise, the engine executes the service related to the *raise blinds* task because its context precondition is the opposite one (*OutsideBrightness!=low*).

Finally, the engine gets the last task. This is related to the previous task by the >>[UserPresence=Kitchen]>> relationship; therefore, the engine waits until Bob enters in the kitchen and then executes the makeCoffee service.

B. Implementation Details

To describe the task model, we have developed a graphical editor using the Eclipse platform, and the EMF and GMF plugins. By using this editor, the model can be graphically edited as shown in Figure 2. These descriptions are stored in XMI (XML Metadata Interchange), which is machine-interpretable at runtime. The context repository is represented as an OWL (Web Ontology Language) ontological model. OWL is an ontology markup language W3C standard that greatly facilitates runtime interpretation and reasoning.

The *context manager* and the *automation engine* are implemented in Java/OSGi technology and are run in an OSGi server together with the pervasive services. Note that the infrastructure is decoupled from the service implementation since we only need to indicate a service identifier.

Using OSGi, the context manager can listen to the changes produced in the services to detect context changes and can also inform the engine when a change is detected. To execute a task, the engine searches for the pervasive service associated to the task in the OSGi server by using its service registry. Then, the engine executes the corresponding service by using the Java Reflection capabilities.

To manage the task model at runtime, the engine uses the EMF Model Query plugin that allows a system to work with any model by querying its structure at runtime. To manage the context repository at runtime, the context manager uses the OWL API 2.1.1, which provides facilities for creating, examining, and modifying an OWL model; and the Pellet reasoner 1.5.2., which allows the OWL model to be queried.

More technical details can be found in [13].

V. Validation of the Proposal

In order to validate the presented software infrastructure, we have applied a case-study based evaluation by following the research methodology practices provided in [14].

The purpose of the evaluation was to validate that our software infrastructure supports the execution of adaptive routines and only automates the routines that users want and in the way they want them. To validate this, we evaluated the following research questions according to the requirements presented in Section III:

 ISBN: 978-1-61208-274-5

1) Does the infrastructure facilitate user participation to take into account user automation desires and demands?
2) Does the infrastructure correctly automate the routines in a context-adaptive way?
3) Does the infrastructure allow routines to be adapted by changing the task model at runtime?

We now summarize the results of this evaluation. More details can be found in [13]. Also, we have recorded several videos that show the software infrastructure in execution. They can be found at http://www.pros.upv.es/art.

A. User Participation

We designed and developed 14 case studies in the smart home domain, covering different set of inhabitants (families, couples and single people). We selected smart homes because this a is fertile ground for offering products and services to improve people' lives. Specifically, the overall purpose of the developed case studies was to make inhabitants' lives more efficient and comfortable and to save energy consumption. A total of 18 subjects between 26 and 57 years old participated as the clients of the case studies (8 female and 10 male). Ten of them had a strong background in computer science, while the rest only had basic computer knowledge.

We identified from 6 to 12 routines to be automated in each case study resulting in a total of 97 routines. It took us between 40 and 90 minutes to specify the routines of each case study using the task model. We then briefly taught the subjects about the main components of the task model notation and evaluated their comprehension to determine if the task model facilitated user participation. To do this, we used a short semi-structured interview in which we asked the subjects questions to make them reason about the task model. For instance, some of these questions were: how many tasks will be executed in this routine?; when will this routine be activated?; when will this task be executed?.

We found that 14 of the 18 subjects understood the routines specified in the task model perfectly. The other 4 users, those with little mathematics and computer skills, understood the structure of the model (task hierarchy and task relationships) very well; however, they had difficulty knowing what the used context conditions meant. To solve this problem, we added a new view in the task model editor to show these conditions in natural language. For instance, instead of showing *[OutsideBrightness=low] switch lights on* in the model, we show: *if the outside brightness is low, switch lights on*, or instead of showing $>> [UserPresence = Kitchen] >>$, we show *when you arrive to the kitchen*.

After checking the subjects' comprehension of the model, we explained the specified routines to them. We found that the task model is very useful in discussing and validating the routines to be automated. If something was not specified the way the users wanted it to be automated, we refined the model to fulfil their requirements. We repeated this process until the users agreed with the specification. This allowed us to describe the routines by taking into account the automation desires and demands of the users.

B. Context-Adaptive Routine Automation

Once the task models were validated, we put the system into operation to automate the described routines. We used a scale environment with real devices (see http://pros.upv.es/art) to represent the Smart Home. This execution environment was made up of a PC and a network of KNX devices connected to the PC by a USB port. An Equinox distribution (which is the OSGi implementation of Eclipse) was run in the PC. The software infrastructure together with the pervasive services required to execute the leaf tasks of the routines (a total of 26 services) were installed and started in Equinox.

By using JUnit tests, we validated that the routines were correcly automated in a context-adaptive way. Specifically, the following aspects were validated:

- All the routines were triggered only when its context situation was fulfilled.
- When a routine was executed, all the required services were executed in the correct order and in the correct context conditions.

The proposed validation consisted in: (1) simulating the fulfilment of specific context conditions in order to trigger the execution of several routines, and (2) checking that all the services that must be executed were registered by the context manager in the correct order, respecting the corresponding temporal relationships between the tasks.

We performed this process in an iterative way, which allowed us to detect and solve some mistakes. For instance, we realized that the routines dependent on time, made the system enter into a loop. This was because the system updated time every second and the smallest time unit considered in the routines was minutes. Thus, the context situation of these routines was continuously fulfilled until a minute went by. To solve this problem without overloading the system, we updated the context manager to update time every minute.

C. Routine Adaptation after System Deployment

We validated that the routines could be easily evolved by changing the task model at runtime. Specifically, we changed the task model to perform the following types of adaptation: delete routines; modify routines by changing their context situation and their tasks (task order, context preconditions, temporal relationships, etc.); and add new routines.

After each adaptation, we simulated the fulfilment of the context situations of the routines and applied the JUnit tests again to check that the routines were correctly executed according to the performed evolution. For instance, we modified the *WakingUp* routine. We changed the second task in order to wake Bob up with relaxing music; we removed the lighting task, and added a new task so that the system informed Bob about the weather when he was in the kitchen. Figure 4 shows these modifications in the task model and the execution trace of the *WakingUp* routine before and after evolving it.

VI. Conclusion

In this work, we have presented and evaluated a sofware infrastructure that achieves the automation of adaptive daily

 ISBN: 978-1-61208-274-5

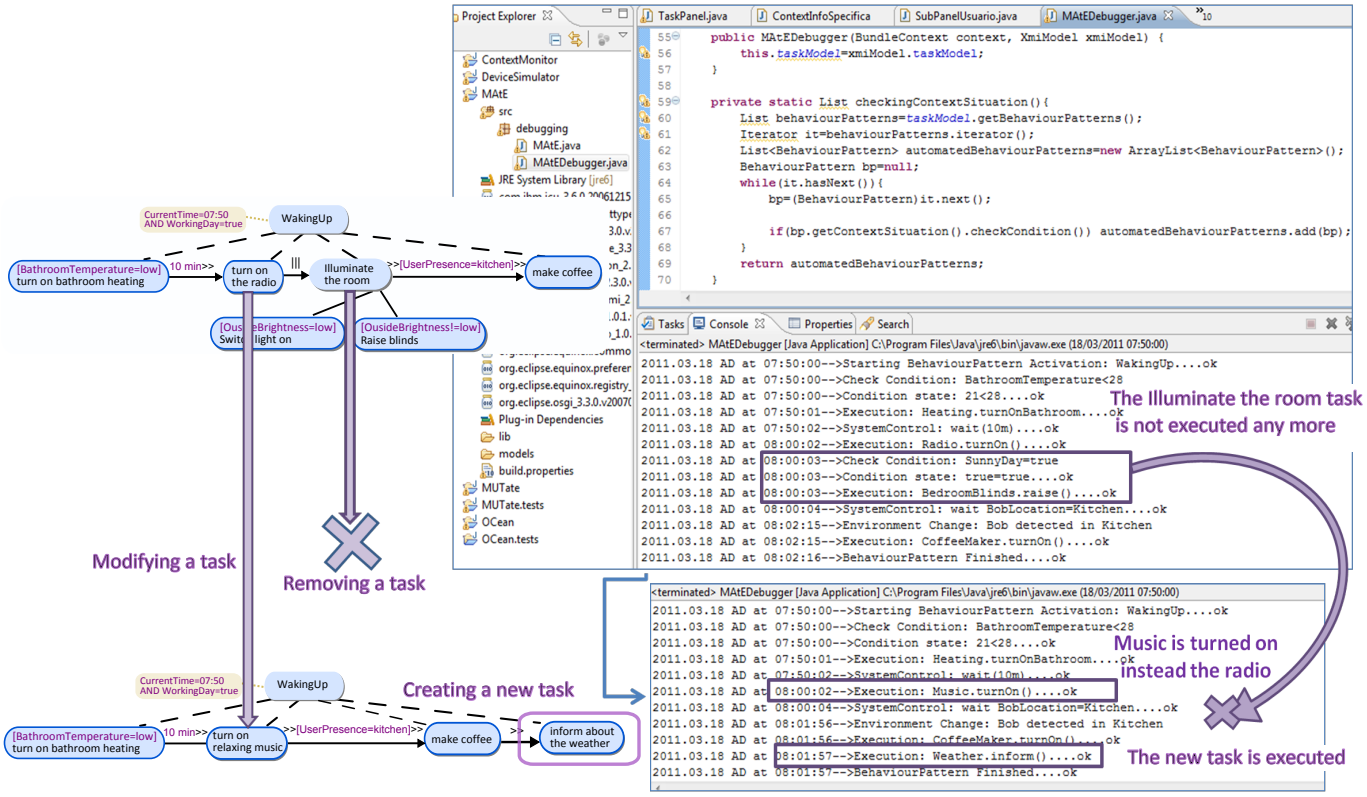


Fig. 4. Execution traces before and after evolving the WakingUp routine

routines. These routines are represented in high-level abstraction context-adaptive models that are directly interpreted at runtime. This considerably facilitates the further adaptation of the routines by changing the models (i.e., at the modelling level) at runtime, which is one of the top challenges in software evolution research [15]. As soon as the models are changed to adapt the routines, the changes are also taken into account by the automation engine.

Further work will be dedicated to extending the approach with machine-learning algorithms in order to provide more automation in the requirements capture and the routine adaptation after system deployment. When the system is running, the context manager stores the user actions in the context repository. Machine-learning algorithms can use this information to detect new routines or changes in the ones already specified and adapt the models accordingly.

ACKNOWLEDGMENT

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

REFERENCES

[1] F. Mattern, "The vision and technical foundations of ubiquitous computing," *Upgrade European Online Magazine*, pp. 5–8, 2001.

[2] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an ambient-intelligence environment using embedded agents," *IEEE Intelligent Systems*, vol. 19, no. 6, pp. 12–20, 2004.

[3] P. Rashidi and D. J. Cook, "Keeping the intelligent environment resident in the loop," in *IE 08*, 2008, pp. 1–9.

[4] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Using context and preferences to implement self-adapting pervasive computing applications," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1307–1330, 2006.

[5] M. García-Herranz, P. Haya, and X. Alamán, "Towards a ubiquitous end-user programming system for smart spaces," *Journal of Universal Computer Science*, vol. 16, no. 12, pp. 1633–1649, 2010.

[6] D. Cook and S. Das, *Smart environments: Technology, protocols and applications*. Wiley-Interscience, 2004, vol. 43.

[7] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, "a cappella: Programming by demonstration of context-aware applications," *CHI 2004*, pp. 33–40, 2004.

[8] J. Chin, V. Callaghan, and G. Clarke, "A programming-by-example approach to customising digital homes," in *IE 08*, 2008, pp. 1–8.

[9] C. Pribeanu, Q. Limbourg, and J. Vanderdonckt, "Task modelling for context-sensitive user interfaces," *Interactive Systems: Design, Specification, and Verification*, pp. 49–68, 2001.

[10] R. Huang, Q. Cao, J. Zhou, D. Sun, and Q. Su, "Context-aware active task discovery for pervasive computing," in *International Conference on Computer Science and Software Engineering*, 2008, pp. 463–466.

[11] F. M. Reyes, "Issues of sensor-based information systems to support parenting in pervasive settings: A case study," *Emerging Pervasive and Ubiquitous Aspects of Information Systems: Cross-Disciplinary Advancements*, p. 261, 2011.

[12] F. Paternò, "From model-based to natural development," *HCI International*, pp. 592–596, 2003.

[13] E. Serral, "Automating routine tasks in smart environments. a context-aware model-driven approach," Ph.D. dissertation, Technical University of Valencia, DSIC, 2011.

[14] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[15] T. Mens, "The ercim working group on software evolution: the past and the future," in *IWPSE-Evol workshops*. ACM, 2009, pp. 1–4.

 ISBN: 978-1-61208-274-5

Self-discovery Algorithms for a Massively-Parallel Computer

Kier J. Dugan, Jeff S. Reeve, Andrew D. Brown
Electronics and Computer Science
University of Southampton
Southampton, UK
{kjd1v07, jsr, adb}@ecs.soton.ac.uk

***Abstract*—SpiNNaker is a biologically-inspired massively-parallel computer design that will contain over a million processors, distributed over more than 60,000 chips. The system bootstrap must discover how they are connected for the machine to enter a usable state. In this paper we describe a set of algorithms for discovering missing or malfunctioning inter-chip links, assigning unique identifiers to each chip, and building point-to-point network routing tables. All of the algorithms have been simulated, and will be implemented into SpiNNaker after further investigation. Our goal is to design an autonomic bootstrap stage that can operate on arbitrary machine geometries.**

***Keywords*—SpiNNaker; self-discovering networks; parallel computer bootstrap procedures; self-configuration.**

I. Introduction

SpiNNaker [1] is a biologically-inspired massively-parallel computer that will contain over a million processors, distributed across more than sixty-thousand Multi-Processor System-on-Chip (MPSoC) devices. The flagship application for this machine is to model large neural-networks containing biologically-realistic numbers of neurons and synapses in biological real-time [2]. Each MPSoC contains 18 ARM processors with the intention of using 16 for simulation, one as a *monitor* processor that manages communications for the chip, and one as a spare for reliability purposes.

The network fabric of SpiNNaker follows a globally asynchronous, locally synchronous (GALS) methodology which allows each processor to exist in its own clock domain [3]. Inside each MPSoC, an asynchronous network-on-chip (NoC) connects all of the processors to the router. These routers communicate with each other using six inter-chip ports which have also been inspired by NoC designs. Both networks use m-of-n codes to provide reliable, low-latency, self-timed communications using compact transmit/receive logic.

Four routing methods, each optimised for a specific task, operate in parallel throughout the machine [4]. MC (multicast) traffic is used to carry address-event representation (AER) simulation data in a one-to-many fashion inspired by neural connectivity patterns; FR (fixed-route) packets are a specialisation of this, where the source-addressed routing has been sacrificed in favour of a larger payload. P2P (point-to-point) packets carry command and system information between two chips of the machine in a one-to-one mapping. Finally, NN (nearest-neighbour) packets provide a one-to-one link between a chip and any one of its six immediate neighbours.

An ideal SpiNNaker network is an isotropic 3D torus with extra diagonal links to facilitate triangular routing around problematic links. Due to the scale of the final machine, there can be no guarantee that all processors and chips will be functional on start-up. Assigning labels and routes statically is therefore not viable, nor can any assumptions be made about the regularity of the structure of the machine.

Other MPSoCs avoid this issue by taking more self-contained approach, acting as either master- or co-processors. In the Centip3De [5] MPSoC, a 3D NoC is used to maintain cache-coherency throughout the chip so that all 64 ARM Cortex-M3 processors can communicate using shared memory. A similar approach has been used by Intel in their prototype data-center-on-a-die, which reserves a small region of shared memory as a *message-passing buffer* that allows the 48 Pentium-class IA-32 processors to communicate [6]. TILE64™ [7] uses several software-controlled networks (one also being software-routed) to connect a regular grid of 64 VLIW processors to the system RAM, on-chip peripherals, and each other. PCI-express and Ethernet controllers provide the inter-chip communications instead of allowing the processor network to bridge chip boundaries as it does in SpiNNaker.

These devices either assign processor labels statically or are structured such that they can be derived at runtime. Similarly, conventional cluster machines assembled from commodity computer hardware may make use of the vendor-assigned MAC address of the network interface card as a machine label. Higher level protocols, such as the Dynamic Host Configuration Protocol (DHCP), can be used to automatically assign system-wide labels from a central source.

SpiNNaker chips are, nominally, identical and hence there is no equivalent of a MAC address available. It follows that only NN packets may be used during the system boot because the higher-level networks require both chip labels and at least partial knowledge of the machine geometry. In this paper we present a set of algorithms that will discover missing/malfunctioning inter-chip links, assign each chip a unique label, and then build the P2P tables. All of these algorithms have been prototyped in simulations that mimic the distributed interrupt-driven nature of SpiNNaker.

The rest of this paper is structured as follows: in Section II

 ISBN: 978-1-61208-274-5

we introduce the bootstrap algorithms that eventually lead to a constructed P2P table on each chip; Section III briefly describes some early-stage work towards building the MC tables; and Section IV concludes with a summary of this paper and introduces planned future work.

II. Bootstrapping Algorithms

The SpiNNaker machine does not power up with any implied knowledge of its structure and must discover this as part of the bootstrap procedure. Each chip has an integrated ROM that stores a small boot program capable of initialising the NN routing mechanism and the Ethernet controller if there is an active connection. All cores enter the wait-for-interrupt (WFI) state once these resources are ready, and may only process interrupts from these sub-systems.

System software, which must be loaded into the machine from an external host, is then distributed to all chips using a flood-fill mechanism [8]. Existing algorithms for providing each SpiNNaker chip with a unique label and building the point-to-point tables are semi-automatic and require prior knowledge. The algorithms presented in this paper aim to remove this constraint and hence provide a more autonomic self-discovering bootstrap process that may be applied to *arbitrary* machine geometries.

A. The α-ping

Malfunctioning links are not detected during the initial flood-fill process primarily due to the small size of the boot ROM. After control is passed from the boot-loader to the system software, higher-level detection algorithms may be applied to the machine to detect faults. Completion of the flood-fill cannot easily be detected without making assumptions about the machine geometry. We propose the α-ping as a process that will be incorporated into the system software and then executed after an appropriate time-out to allow for completion of the flood-fill.

Two tokens are passed between the monitor processors of adjacent chips—the request token, $\alpha_{\mathbf{R}}$, and the acknowledgement token, $\alpha_{\mathbf{A}}$. Each chip labels all local ports as `undefined` immediately after executing the system software. The host machine starts the process by injecting $\alpha_{\mathbf{R}}$ into the Ethernet-connected chip. $\alpha_{\mathbf{R}}$ is then broadcast to all neighbouring chips and every local port is assigned the `requested` label. Chips respond to incoming $\alpha_{\mathbf{R}}$ with $\alpha_{\mathbf{A}}$ and label the appropriate port as `active`; further $\alpha_{\mathbf{R}}$ are broadcast to all other unlabelled ports and the `requested` label will be attached as before. After a predetermined chip-local time-out (for the same reasons as with the flood-fill) all ports that are still labelled `requested` are assumed to have malfunctioned and will hence be `inactive`.

B. Assigning Chip Labels

Each chip of the SpiNNaker machine must be assigned a unique identifier so that P2P routes can be established. The existing method assumes a grid topology and requires extents in X and Y to be specified *a priori* by the operator [9]. A chip will be assigned a label from a predecessor (which is an Ethernet-connected host in the case of the root chip) and then geometric assumptions and simple arithmetic ($x \bmod X$ and $y \bmod Y$) are used to calculate the labels for the surrounding chips. Two clear pros of this method are that a) chip labels are entirely deterministic so there cannot be any conflicts during the set-up; and b) it will operate as a wave front of parallel computation emanating from the root chip. However, the geometric assumptions constrain the SpiNNaker machine to a grid which may not always be an appropriate geometry.

A solution is to build a spanning tree with its root at the Ethernet-connected chip. The generated tree structure may be derived from an *arbitrary* connected graph and provides a simple method for building hierarchical barrier, scatter and gather constructs common in parallel computing. Chips can also be uniquely labelled as part of the traversal process that builds the tree.

The SpiNNaker programming model is based on *events* (i.e., hardware interrupts) that are triggered either by a regular timer tick or by a packet arrival. Deriving the spanning tree must be performed within SpiNNaker and can only make use of NN packets to raise events on neighbouring chips. Only the monitor processors can take part in this process, and the algorithm must be defined on a per-node basis rather than on the machine-graph as a whole.

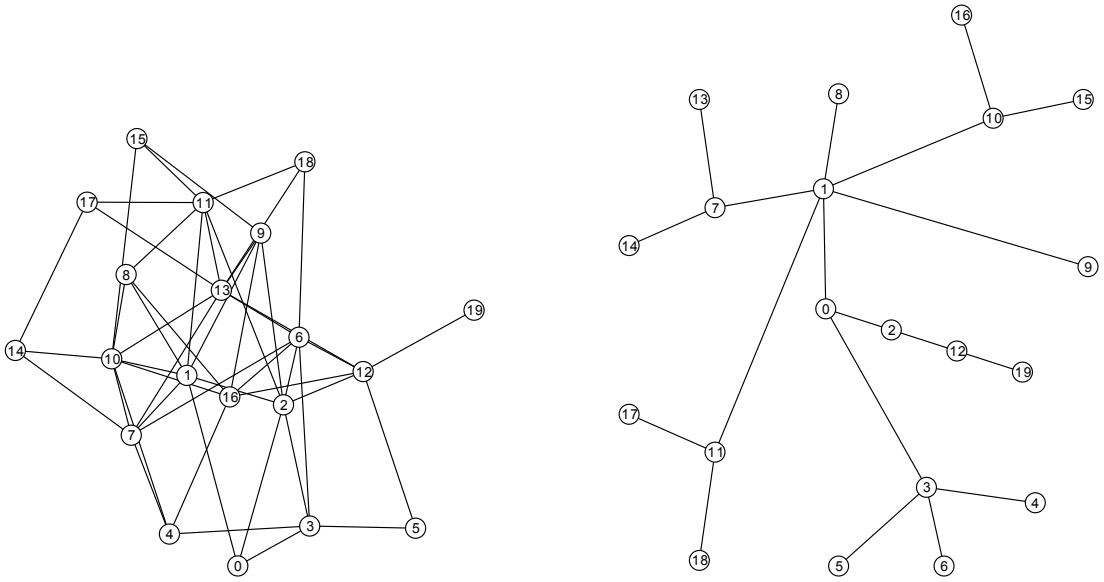
An interrupt-driven breadth-first search (BFS) is used because it should extract a wide, shallow tree from the SpiNNaker grid. This is desirable as it allows a large volume of the barrier, scatter and gather communication to occur in parallel. A sequential BFS is presently used because it ensures that labels are generated contiguously and will therefore be unique across the machine as illustrated by Figure 1.

1) General Algorithm Description: A node is represented as a finite-state machine and will be in one of the following states during the algorithm: `IDLE`, `LABELLED`, `PARENT`, or `BARRIER`. All nodes begin in the `IDLE` state and enter the `BARRIER` state once the tree has been successfully built. Query-events, $Q(L)$, and reply-events, $R(L, A)$, are used to transmit labels, L, between parent and child nodes and to report the number of nodes affected by an operation, A.

For a given node, V, in the `IDLE` state, a label will be attached upon the reception of an event, $Q(L)$, containing the new label value to use. V will then emit a reply-event, $R(L, 1)$, to the originator of Q to report that the label has been accepted (i.e., $A = 1$ because V was the only node affected by Q). V will then advance to the `LABELLED` state and hence a parent-child relationship has been established.

A node in the `LABELLED` state will receive an event, $Q(L)$, after the parent, V_P, has finished labelling its neighbour nodes. L will be the first value that may be used as a label. V will iterate over all neighbouring nodes (except V_P) sending $Q(L+n)$ where n is initialised to 0 and incremented for each $R(L+n, 1)$ reply. Neighbours that already have been assigned a label will respond with $R(L+n, 0)$ and hence n will not be incremented. This process will continue until all neighbours of V have been visited, causing V to respond to V_P with

 ISBN: 978-1-61208-274-5



(a) Arbitrary machine graph (b) Extracted tree

Fig. 1. Input graph and resulting tree for the proposed event-driven breadth-first algorithm.

$R(L+n, n)$ before entering the PARENT state. Using Figure 1 as an example and assuming that node 0 is in the LABELLED state (i.e., the host machine has assigned it a label of 0), it will pass $Q(1)$ to one of its neighbours which will then respond with $R(1, 1)$ to indicate that the label, 1, has been accepted. If the target neighbour already has a label, the reply will instead be $R(1, 0)$ because the label has not been accepted. The next neighbour receives $Q(2)$ and will reply with $R(2, 1)$, and so on until all neighbours have been labelled.

Once in this state, V performs similar behaviour but instead computes a running total of affected nodes, n_T, which is first initialised to 0 and then increased by n_R for each reply event $R(L + n_T + n_R, n_R)$ from a child node. V will respond to its parent, V_P, with $R(L + n_T, n_T)$ after all child nodes have been visited. Following from the previous example, node 0 would respond to its parent (the host, in this case) with $R(3, 3)$ because $L = 3$ was the highest label assigned to a neighbour and $A = 3$ nodes were affected.

A node may, at any time, receive a barrier event, $B(L)$, which immediately causes the node to perform the following actions: 1) store L as the number of nodes in the machine graph, 2) propagate $B(L)$ to all child nodes, and 3) transition into the BARRIER state. No BFS events will be processed in this state, hence this marks the completion of the algorithm.

2) Duty of the Root Node: The description in the previous section is valid for all nodes of the graph and of the derived tree, but the root node is required to behave slightly differently. Its parent is the *host* of the simulation and will not be part of the machine graph (an Ethernet-connected PC is the host of a SpiNNaker-based simulation). The host will issue $Q(0)$ to a node V_R of the target system to start the algorithm. V_R will assert itself as the root node of the machine because $L = 0$.

V_R will progress through the states in the same manner as any other node except that it does not require the permission of its parent to raise new events. It therefore issues new events, calculating appropriate values for L, and computes a running total of the number of affected nodes, A_R, for each pass. When $A_R = 0$, all nodes of the machine graph have been assigned a unique label and have progressed through all the required states, which triggers V_R to perform the following:

- $B(L_{max})$ is issued to all child nodes of V_R;
- $R(L_{max}, 0)$ is issued to the *host* of the simulation;
- V_R enters the BARRIER state.

Following the running example one final time with node 0 having labelled its neighbouring nodes and entered the PARENT state, a random child is chosen (node 3 in the case of Figure 1) and issued with $Q(4)$. Node 3 follows the same procedure as before by labelling its neighbours 4, 5 and 6. Once complete, node 3 replies to node 0 with $R(6, 3)$ because $L = 6$ is the highest label used and $A = 3$ nodes were affected. Next, node 1 is randomly chosen and issued $Q(7)$; $R(11, 5)$ is raised in response after each neighbour has been visited. Finally node 2 is issued $Q(12)$ and responds with $R(12, 1)$. Node 0 can now calculate the total number of affected nodes for this pass as $A_R = 3 + 5 + 1 = 9$. As $A_R \neq 0$, a random child node is passed $Q(13)$ and the process continues until nodes 1, 2, and 3 all respond with $R(L_{max}, 0)$, which causes node 0 to complete the actions described above. Figure 1a shows a random graph that has been used as a machine model for this algorithm. Each node represents a SpiNNaker chip and each edge is a nearest-neighbour (NN) connection. Figure 1b is the tree structure that has been built.

C. P2P Table Generation

Building the P2P routing tables currently uses the same machine geometry assumptions as the labelling process [9],

and must be replaced to use the BFS-assigned chip labels.

Assuming that all nodes of an arbitrary connected graph (i.e., all chips of a SpiNNaker machine) are in the BARRIER state, a node, V_i, transmits its label, i, to *all* neighbouring nodes including those that are not child nodes. A receiving node, V_k, with network ports $E(V_k)$ links the incoming port, $E_j \in E(V_k)$, to V_i by setting the ith entry of its routing table accordingly (i.e., $P_k[i] = E_j$). The message is then forwarded through all ports other than E_j to continue the process.

This bootstrap stage begins with the root node broadcasting its own label, 0, to its neighbours after all graph nodes have entered the BARRIER state. Nodes receiving label messages for the *first* time update their P2P table, propagate the message to all neighbours except the source, and then broadcast their own label. If a P2P table entry is already present then the node will not broadcast further messages. A wave front of these messages will propagate across the graph until all nodes have a complete P2P table, which will contain L entries as reported by the $B(L)$ message of the labelling stage.

A second barrier condition is required to conclude the bootstrap. Messages may now be routed between any two nodes of the machine graph dynamically by following the appropriate ports mapped in the P2P tables.

III. Mapping Problem Graphs to Machine Geometries

A SpiNNaker application is represented as a connected graph that describes how data flows through a set of behaviours. Mapping these *problem graphs* onto the machine is essentially a combination of assignment and path-finding problems. SpiNNaker is optimised for simulating large-scale neural networks in biological real-time, and hence existing methods exploit the hierarchy of these problem graphs to simplify allocation and routing [10][11].

These assumptions do not hold for general-purpose applications because there can be no guarantee of the structure of the problem graph. We are developing a physically-inspired approach that treats each node of the graph as a charged particle contained within a volume. The field interactions between nodes will distribute them evenly across the machine geometry. Additional forces acting in place of the edges will keep heavily connected areas local. Graph drawing and chip-layout algorithms have served as two key inspirations.

Our goal is to produce an algorithm that can be solved locally at each node without requiring any global knowledge of the machine or problem graphs. This cannot be included in the bootstrap because the application may change during runtime. An ongoing supervisory process may be able to adjust node and edge weights (i.e., their field contributions) to facilitate dynamic load balancing without requiring global knowledge.

IV. Conclusion and Future Work

We have presented a brief review of bootstrapping algorithms that we are developing for use on the SpiNNaker massively parallel computer. A breadth-first search based on node-local information and event driven interactions is used to assign unique labels to nodes (chips) and to support a barrier tree structure. At present, all algorithms have been tested in a simulation environment that accurately mimics the NN network. The BFS is sequential to guarantee unique labels across the machine but this leads to a computational complexity of $O(n)$. This is somewhat wasteful of the massively parallel resources of SpiNNaker, and further work will be conducted to parallelise this algorithm as much as is practicable.

Section III presents an idea we aim to develop that will allow arbitrary problem graphs to be mapped onto arbitrary machine geometries using only locally available knowledge. Our longer term goal is to couple these algorithms to design a system capable of reacting to system-level changes (such as a processor or chip malfunctioning) without using global knowledge or a central overseer. Additionally, they will allow a problem graph to be streamed into SpiNNaker and the allocation of problem nodes to cores, and the derivation of network routes, will be an automatic process.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grants EP/G015775/1 and EP/G015740/1, and with industrial support from ARM Ltd.

References

[1] SpiNNaker home page. University of Manchester. Last Accessed: May 2013. [Online]. Available: http://apt.cs.man.ac.uk/projects/SpiNNaker/

[2] S. Furber and A. Brown, "Biologically-Inspired Massively-Parallel Architectures - Computing Beyond a Million Processors," in *Int. Conf. on Application of Concurrency to System Design*. IEEE, 2009, pp. 3–12.

[3] L. Plana, S. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS Infrastructure for a Massively Parallel Multiprocessor," *Design & Test of Computers*, vol. 24, no. 5, pp. 454–463, Sep. 2007.

[4] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, pp. 1–14, 2012.

[5] D. Fick, R. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Sylvester, and D. Blaauw, "Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores," in *ISSCC*. IEEE, Feb. 2012, pp. 190–192.

[6] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, and Others, "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *ISSCC*, vol. 9, no. 2. IEEE, Feb. 2010, pp. 108–109.

[7] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-c. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64? Processor: A 64-Core SoC with Mesh Interconnect," in *ISSCC*. IEEE, Feb. 2008, pp. 88–89.

[8] M. Khan, J. Navaridas, A. Rast, X. Jin, L. Plana, M. Lujan, J. Woods, J. Miguel-Alonso, and S. Furber, "Event-Driven Configuration of a Neural Network CMP System over a Homogeneous Interconnect Fabric," in *8th Int. Symp. on Parallel and Distributed Computing*. IEEE, 2009, pp. 54–61.

[9] T. Sharp, C. Patterson, and S. Furber, "Distributed configuration of massively-parallel simulation on SpiNNaker neuromorphic hardware," in *Int. Joint Conf. on Neural Networks*. IEEE, 2011, pp. 1099–1105.

[10] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Computing Frontiers*. ACM Press, 2012, pp. 183–192.

[11] S. Davies, J. Navaridas, F. Galluppi, and S. Furber, "Population-based routing in the SpiNNaker neuromorphic architecture," in *Int. Joint Conf. on Neural Networks*. IEEE, 2012, pp. 1–8.

 ISBN: 978-1-61208-274-5

A Software Design Pattern Based Approach to Adaptive Video Games

Muhammad Iftekher Chowdhury, Michael Katchabaw
Department of Computer Science
University of Western Ontario
London, Canada
{iftekher.chowdhury, katchab}@uwo.ca

Abstract—**To achieve success, it is becoming increasingly clear that modern video games must be adaptive in nature – malleable and able to reshape to the needs, expectations, and preferences of the player. Failure to adapt results in a game that is too inflexible, rigid, and pre-defined; one that is simply ineffective, particularly for a large and diverse player population. Developing and supporting adaptive games, however, introduces many challenges. In this paper, we describe a set of software design patterns for enabling adaptivity in video games to address these challenges. We also demonstrate the benefits of our pattern-based approach, in terms of software quality factors and process improvements, through our experience of applying it to a number of video games for enabling a particular type of adaptivity, auto dynamic difficulty.**

Keywords-adaptive video game; software design patterns; game development process; software quality

I. INTRODUCTION

Building rich and dynamic video games is surprisingly complex [1], so much of the existing research and development in this area has led to the creation of games that are largely deterministic in nature. What occurs in these worlds and how this is presented to the player is for the most part fixed, and quite unable to adequately react to the interactions of the player [2,3]. While interesting in their own ways, these games are often too inflexible and rigid to be able to effectively meet the needs and expectations of a large and diverse player population [2,4,5,6], especially as these needs and expectations change as players mature, refine their skills, and form new experiences [7]. In the end, this leads to a loss of engagement, a break of immersion, and an overall disappointing player experience [2,8,9]. The result is a game that is unsuccessful critically and commercially.

As work in this area continues, it is becoming increasingly clear that games must be adaptive in nature — malleable and able to reshape to the needs, expectations, and preferences of the player [2,3]. Adaptive systems are designed to excel at situations that cannot be completely or singularly modeled prior to development, and so they must be able satisfy requirements that arise only after they are put in use; this is very much the case in games. Nearly every aspect of a game can be made adaptive in this way: the game world (structural elements, composition); the population of the world (the agents or characters in the world); any narrative elements (story, history, or back-story); gameplay (challenges, obstacles); the presentation of the game to the player (visuals, music, sound); and so on. In being adaptive, games can provide more compelling, engaging, immersive, and perhaps personalized or customized experiences to their player, leading to a significantly better outcome for the player, and far more success for the game in the end [2,4,5,6,8,9,10].

Previous attempts at adaptivity can be characterized as ad hoc from a software engineering perspective; lacking rigor, structure, and reusability, with custom solutions per game, which is not acceptable [11,12]. There is a critical need for reusable software infrastructure to enable the construction of adaptive games [11,12]. Addressing this problem is the broad goal of our research. While this is a difficult goal to achieve [2,13], both from theoretical and practical perspectives, we have found success in this area by leveraging software design patterns [14].

In particular, we study adaptivity in games through an exploration of a particular problem in this space, that of auto dynamic difficulty. In this case, adaptations are focused on adjusting game difficulty to match the expertise of the player. According to the theory of flow or optimal experience [15], players who lack the skill to suitably deal with the challenges they face will feel anxiety or frustration in their experience, while players whose skills are excessive for the challenges faced will feel boredom or receive no sense of accomplishment from their experience. A game that is properly balanced, on the other hand, will be much better received by the player [16]. A single difficulty level has little chance of addressing the needs of a broad audience. Multiple static difficulty levels in games also fail in this context, as they expect the players to judge their ability themselves appropriately before playing the game and also try to classify them in broad clusters [11,12]. An adaptive game supporting auto dynamic difficulty circumvents these problems to deliver a more satisfying experience to players by providing per-player skill-appropriate challenge.

In this paper, we discuss our general approach to adaptive games and demonstrate the effectiveness of our approach by examining auto dynamic difficulty, extending our previous work in this area [11,12]. To do so, we leverage the benefits of software design patterns, derived from self-adaptive system literature [17], to construct an adaptive system for video games that is reusable, portable, flexible, and maintainable.

 ISBN: 978-1-61208-274-5

II. RELATED WORK

In recent years, adaptive video games and auto dynamic difficulty have received notable attention from numerous researchers. In the subsections below, we review key work in this area and discuss the research gap that remains.

A. Adaptive Game Systems

The study of adaptive systems in a broader sense is not new. Unfortunately, it is difficult to directly apply adaptive systems work from other domains to video games [11,12]. Games do more than deliver functionality as in other software systems; there is a larger emphasis on engagement, immersion, and experience, as well as greater demands on interactivity and real-time performance and presence. These factors require careful consideration often not required in other domains. Furthermore, adaptations in games can go beyond the tuning found in most other domains; there can also be creative or generative aspects to adaptivity. There exists a separation of logic or processing and content in games; while both can be tuned, the content aspect can be altered in fundamentally different ways that fall outside of traditional approaches to adaptive systems. Consequently, there is a need to study adaptivity in the context of games. To date, efforts in doing so have been rather scant, with the work of Charles et al. [5] one of the few examples. Unfortunately, attempts in this area tend not to leverage progress from the adaptive systems literature, and so are typically too narrow, overly focused, and lack rigor from a software engineering perspective.

That said, while not studying adaptivity in games directly, many researchers studying other issues in this space have created work that has been adaptive, at least to a certain degree. This includes work on agent and story adaptation [18,19,20,21,22,23], varying the structure of the game world [10,24,25,26], and difficulty adjustment, as discussed at length in the next section. Unfortunately, this work is also quite ad hoc and cannot be readily generalized or reused for other purposes.

B. Auto Dynamic Difficulty

There have been numerous attempts made towards providing auto dynamic difficulty in video games over the years. In this section, we highlight several of these works.

Bailey and Katchabaw [16] developed an experimental testbed based on Epic's Unreal engine that can be used to implement and study auto dynamic difficulty in games. A number of mini-game gameplay scenarios were developed in the test-bed and these were used in preliminary experiments.

Rani et al. [27] suggested a method to use real time feedback, by measuring the anxiety level of the player using wearable biofeedback sensors, to modify game difficulty. They conducted an experiment on a Pong-like game to show that physiological feedback-based difficulty levels were more effective than performance feedback to provide an appropriate level of challenge. Physiological signals data were collected from 15 participants each spending 6 hours in cognitive tasks (i.e., anagram and Pong tasks) and these were analyzed offline to train the system.

Hunicke [28] used a probabilistic model to design adaptability in a first person shooter (FPS) game based on the Half Life SDK. They used the game in an experiment on 20 subjects and found that adaptive adjustment increased the player's performance (i.e., the mean number of deaths decreased from 6.4 to 4 in the first 15 minutes of play) and that players did not notice the adjustments.

Hao et al. [29] proposed a Monte-Carlo Tree Search (MCTS) based algorithm for auto dynamic difficulty to generate intelligence of Non Player Characters (NPCs). Because of the computational intensiveness of the approach, they also provided an alternative based on artificial neural networks (ANN) created from the MCTS. They also tested the feasibility of their approach using Pac-Man.

Hocine and Gouaïch [30] described an adaptive approach for pointing tasks in therapeutic games. They introduced a motivation model based on job satisfaction and activation theory to adapt task difficulty. They also conducted preliminary validation through a control experiment on eight healthy participants using a Wii balance board game.

C. Research Gap

It is clear from surveying the literature that a structured, formalized study of adaptivity for video games is needed to continue advancing the state of the art in this area. Indeed, games could benefit greatly by having an infrastructure of frameworks, patterns, libraries, and support tools to enable adaptivity, as is the focus of this paper. In doing so, developers can focus on creating their games and choosing the adaptations desired, leaving the implementation of these adaptations to the provided infrastructure.

Research on auto dynamic difficulty in games focuses on tool building (including frameworks, algorithms, and so on) and empirical studies, but they all use an ad hoc approach from a software engineering perspective. Thus, in this paper, we discuss a software design patterns based approach for enabling adaptivity in games, and explore the application of this approach to auto dynamic difficulty in particular.

III. DESIGN PATTERNS FOR ADAPTIVE GAMES

In this section, we overview our collection of four design patterns for enabling adaptivity in video games. These patterns were derived from the self-adaptive system literature [17], and specialized and refined for games in particular. For further details, the reader is encouraged to refer to [11] for elaborated discussion and examples.

A. Sensor Factory

The sensor factory pattern is used to provide a systematic way of collecting data on a game and its players while satisfying resource constraints, and provide those data to the rest of the adaptive system. *Sensor* (please see Figure 1) is an abstract class that encapsulates the periodical collection and notification mechanism. A concrete sensor realizes the Sensor and defines specific data collection and calculations. The *SensorFactory* class uses the "factory method" pattern

 ISBN: 978-1-61208-274-5

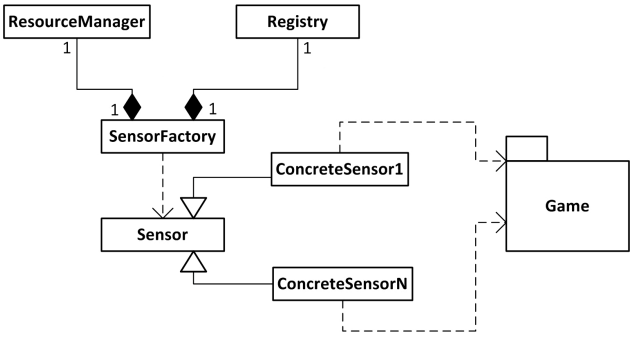


Figure 1. *Sensor factory* design pattern

to provide a unified way of creating any sensors. It takes the *sensorName* and the *object* to be monitored as input and creates the sensor. Before creating a sensor, the *SensorFactory* checks in the *Registry* data structure to see whether the sensor has already been created. If created, the *SensorFactory* just returns that sensor instead of creating a new one. Otherwise, it verifies with a *ResourceManager* whether a new sensor can be created without violating any resource constraints.

B. Adaptation Detector

With the help of the sensor factory pattern, the *AdaptationDetector* (please see Figure 2) deploys a number of sensors in the game and attaches observers to each sensor. *Observer* encapsulates the data collected from sensor, the unit of data (i.e., the degree of precision necessary for each particular type of sensor data), and whether the data is up-to-date or not. *AdaptationDetector* periodically compares the updated values found from *Observer*s with specific *Threshold* values with the help of the *ThresholdAnalyzer*. Each *Threshold* contains one or more boundary values as well as the type of the boundary (e.g., less than, greater than, not equal to, etc.). Once the *ThresholdAnalyzer* indicates a situation when adaptation might be needed, the *AdaptationDetector* creates a *Trigger* with the information that the rest of the adaptation process might need.

C. Case Based Reasoning

While the adaptation detector determines the situation when an adjustment is required by creating a *Trigger*, case based reasoning (please see Figure 3) formulates the *Decision* that contains the adjustment plan. The

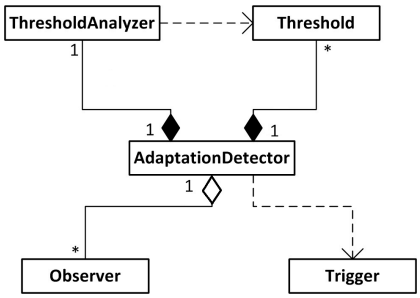


Figure 2. *Adaptation detector* design pattern

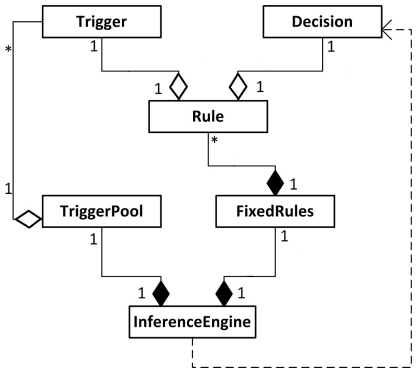


Figure 3. *Case based reasoning* design pattern

InferenceEngine has two data structures: the *TriggerPool* and the *FixedRules*. *FixedRules* contains a number of *Rule*s. Each *Rule* is a combination of a *Trigger* and a *Decision*. The *Trigger*s created by the adaptation detector are stored in the *TriggerPool*. To address the triggers in the sequence they were raised in, the *TriggerPool* should be a FIFO data structure. The *FixedRules* data structure should support search functionality so that when the *InferenceEngine* takes a *Trigger* from the *TriggerPool*, it can scan through the *Rule*s held by *FixedRules* and find a *Decision* that appropriately responds to the *Trigger*.

D. Game Reconfiguration

Once the adaptive system detects that an adjustment is necessary, and decides what and how to adjust the various game components, it is the task of the game reconfiguration pattern (please see Figure 4) to facilitate smooth execution of the decision. The *AdaptationDriver* receives a *Decision* selected by the *InferenceEngine* (please see case based reasoning in previous subsection) and executes it with the help of the *Driver*. *Driver* implements the algorithm to make any attribute change in an object that implements the *State* interface (i.e., that the object can be in ACTIVE, BEING_ACTIVE, BEING_INACTIVE or INACTIVE states, and outside objects can request state changes). As the name suggests, in the active state, the object shows its usual behaviour whereas in the inactive state, the object stops its regular tasks and is open to changes.

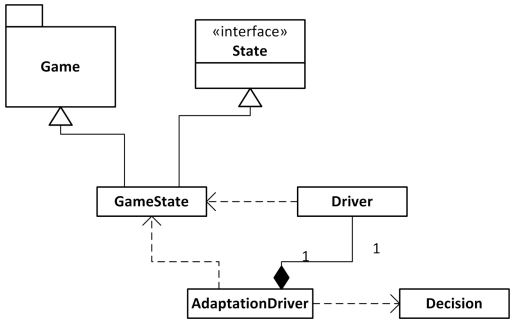


Figure 4. *Game reconfiguration* design pattern

 ISBN: 978-1-61208-274-5

The *Driver* takes the object to be reconfigured (default object used if not specified), the attribute path (i.e., the attribute that needs to be changed, specified according to a predefined protocol such as object oriented dot notation) and the changed attribute value as inputs. The *Driver* requests the object that needs to be reconfigured to be inactive and waits for the inactivation. When the object becomes inactive, it reconfigures the object as specified. After that, it requests the object to be active and informs the *AdaptationDriver* when the object becomes active. The *GameState* maintains a *RequestBuffer* data structure to temporarily store the inputs received during the inactive state of the game. (If the reconfiguration is done efficiently, however, it should be completed within a single tick of the main game loop, and this buffering should be largely unnecessary.) The *GameState* overrides Game's event handling methods and game loop to implement the *State* interface.

E. Integration of Design Patterns

In [31], Salehie and Tahvildari described integration of four generic steps for an adaptation process namely monitoring, detecting, deciding, and acting. The four design patterns discussed in previous sections work on the same process flow. In this Section, we briefly re-discuss how they work together to create a complete adaptive system (please see Figure 5). The sensor factory pattern uses Sensors to collect data from the game so that the player's state and the game's state can be measured. The adaptation detector pattern observes Sensor data using Observers. When the adaptation detector finds situations where the game needs to be adjusted, because either the player or the game is in a sub-optimal state, it creates Triggers with appropriate additional information. Case based reasoning is then notified about required adjustments by means of Triggers. It finds appropriate Decisions associated with the Triggers and passes them to the adaptation driver. The adaptation driver applies the changes specified by each Decision to the game, to adjust the functioning of the game accordingly, with the help of the Driver. The adaptation driver also makes sure that the change process is transparent to the player. In this way, all four design patterns work together to create a complete adaptive system for a particular game.

F. Enabling Auto Dynamic Difficulty

When used together, these software design patterns are sufficient to implement a wide range of adaptivity in gameplay. To demonstrate their use, we explore the particular adaptation of game challenge delivered to the player in the form of auto dynamic difficulty.

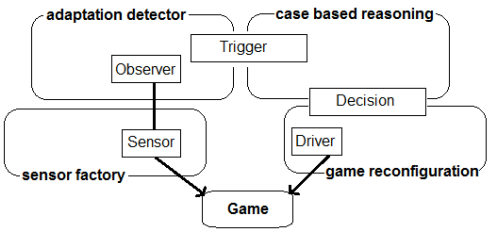


Figure 5. Four design patterns working together in a game

In this application, Sensors would be used to collect data from the game to assess the player's perceived level of difficulty. As above, the adaptation detector pattern observes Sensor data using Observers. When the adaptation detector finds situations where difficulty needs to be adjusted, because the game is currently too easy or too hard for the player, it creates Triggers with appropriate additional information. This information details the in-game activity that gave rise to the Triggers, provides more information on the player's state, and includes anything else needed to assist in formulating a Decision or carrying out reconfiguration. These Triggers are passed to case based reasoning, which in turn finds appropriate Decisions to bring game difficulty back in line with player skill and expertise. These Decisions are then passed to the adaptation driver, which applies the changes specified by each Decision to the game, to adjust the difficulty of the game appropriately, with the help of the Driver. In doing so, the situation is corrected, and game difficulty is tuned according to the needs of the player.

IV. OVERVIEW OF STUDIED GAMES AND ADAPTATIONS

The software design patterns in Section III have been implemented as a Java framework that can be used to enable adaptivity in games. As there is nothing Java-specific to our patterns, bringing this framework to other platforms with other language bindings is part of on-going work.

To date, we have used three very different games developed in Java for studying our approach to adaptivity, with a focus on auto dynamic difficulty. In our earlier work ([11,12]), two casual prototypical games were used. The first game is a variant of Pac-Man and was developed specifically for the purposes of our research. The second game, TileGame, is a slightly modified version of a platform game described in [32]. Even though we were successful in using our approach in these two games, the code for these games was either written by ourselves or well documented and simple enough to be easily understood and reshaped accordingly. Thus, recently we have selected a commercially successful sandbox game – Minecraft [33] to extend our study. Minecraft is commercially available for several platforms, but we focus on the desktop version also developed in Java. In the subsections below, we briefly describe each of the games and examples of adaptations that were implemented using our framework.

A. Pac-Man

In this game, the player controls Pac-Man in a maze (please see Figure 6). There are pellets, power pellets, and 4 ghosts in the maze. Pac-Man has 6 lives. Usually, ghosts are in a predator mode and touching them will cause the loss of one of Pac-Man's lives. When Pac-Man eats a power-pellet, it becomes the predator for a certain amount of time. When Pac-Man is in this predator mode and eats a ghost, the ghost will go back to the center of the maze and will stay there for a certain amount of time. Eating pellets gives points to Pac-Man. The player tries to eat all the pellets in the maze without losing all of Pac-Man's lives. The player is

 ISBN: 978-1-61208-274-5

Figure 6. Screen captured from the Pac-Man game

motivated to chase the ghosts while in predator mode, as that will benefit them by keeping the ghosts away from the maze for a time, allowing Pac-Man to eat pellets more freely. Ghosts only change direction when they reach intersections in the maze, while Pac-Man can change direction at any time. A ghost's vision is limited to a certain number of cells in the maze. Ghosts chase the player if they can see them. If the ghosts do not see Pac-Man, they try to roam the cells with pellets, as Pac-Man needs to eventually visit those areas to collect the pellets. If the ghosts do not see either Pac-Man or pellets, they move in a random fashion.

B. *TileGame*

The level structure and gameplay of this game is similar to the popular Super Mario game series. In this game, the player controls the player character in a platform world (please see Figure 7). There are three levels, each having different tile based maps. Each level is more difficult and lengthier than the previous level, but has more points to give the player a sense of progress and accomplishment.

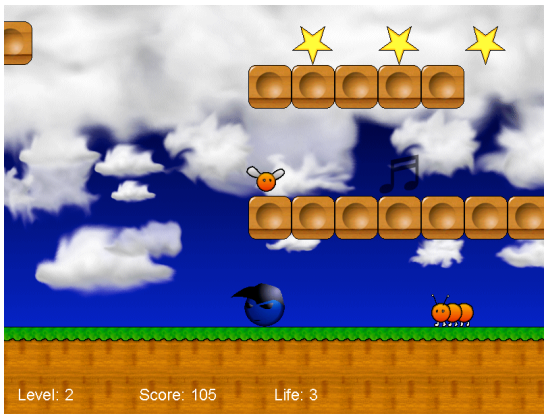


Figure 7. Screen captured from the TileGame game

There are power ups and non-player characters (i.e., enemies) in each level. There are three different types of power ups: basic power ups, bonus power ups, and a goal power up. Basic power ups and bonus power ups give certain points to the player. In each level there is one goal power up that can be found at the end of the level. The goal power up takes the player from one level to another. There are two different types of non-player characters: ants and flies. Ants and flies move in one direction and change direction when blocked by the platforms. The player character can run on and jump from platforms. When the player character jumps on (i.e., collides from above) non-player characters, the non-player character dies. If the player character collides with non-player character in any other direction, then the player character dies instead. The player character has 6 lives. When the player character dies, it loses one life and the game restarts from the beginning of that level. The player character and ants are affected by gravity; flies are not. In this game, three map variants were created for each level. For a particular level, the same objects were placed in the map but positioned slightly differently. One map variant was the default version and other two were easier and harder versions of the default map.

C. *Minecraft*

Minecraft [33] is an exceptionally popular sandbox game that allows players to explore, gather resources, combat, craft and build constructions out of textured cubes in a procedurally generated 3D world. The terrain of the game world, consisting of plains, mountains, forests, caves, and waterways, are composed of rough 3D objects (primarily cubes) representing different materials (for example dirt, stone, tree trunks, water, and so on) and arranged in a fixed grid pattern. Players can break (please see Figure 8) and collect these material blocks and craft these blocks to form other blocks (for example, furnaces, bricks, and stairs) and items (for example sticks, axes, and buckets). Players can place collected or crafted blocks and items elsewhere to build structures. The world is divided into biomes (such as deserts, jungles, and snow fields). The time in the game goes through a day-night cycle every 20 real time minutes.

Figure 8. Screen captured from Minecraft

 ISBN: 978-1-61208-274-5

There are various NPCs known as mobs (including animals, villagers, and hostile creatures). Non hostile animals (such as cows, pigs, chickens, and so on) spawn during the daytime and can be hunted for food and crafting materials. Hostile mobs (such as spiders, zombies, and creepers, a Minecraft-unique creature) spawn during nighttime and in dark areas. There are two primary game modes: creative and survival. In creative mode, players have access to unlimited resources, and are not affected by hunger or environmental or mob damage. On the other hand, in survival mode, players need to collect resources (and craft them) and have both a health bar and a hunger bar that must be managed to stay alive and continue playing. The game also features single player and multiplayer options. For this research, we focused on the single player option played in survival mode.

While Minecraft is not open-source, its source code can be readily obtained through the use of a toolchain [34] provided by an active and extensive modding community that decompiles the game back to its source code. The creators of Minecraft accept this practice while an official modding interface is under development.

D. Adaptations Implemented

In Table I, we provide examples of different adaptations that we have implemented in the above games. The first column shows the name of the game. The next three columns show the details of the adaptations implemented. Please note that these columns: metrics for sensors, attributes for modification, and adaptation scenarios also represent the questions: when to adapt, what to adapt, and how to adapt respectively, which is part of the methodology for eliciting essential requirements for adaptive software [31].

TABLE I. EXAMPLES OF ADAPTATIONS IMPLEMENTED

Game	Metrics for Sensors	Attributes for Modification	Adaptation Scenarios
Pac-Man	Total score, Number of times player dies	Ghost's speed, the ghost's vision length, duration of Pac-Man's predator mode, and so on	Modify ghost's speed, duration of Pac-Man's predator mode and so on based on how the average score per life compared to specific thresholds
TileGame	Current level number, Total score, Number of times player dies	Load different versions of the map where default objects and enemies are placed in slightly different positions	Load different versions of the map when the player character goes to the next level or in the next loading of the same level (such as when the player character dies) based on score and lives lost in last level.
Minecraft	Which day in game, Number of times player dies	Display hints about collecting resources and building shelters	If the player is continuously dying during the first night, give the player some hints to progress through the game to make it easier.
	Number of items of particular materials in player's inventory	Hardness of those particular items	Modify the hardness of a particular resource in the game world as the player's inventory of that particular item changes, making it easier or harder to collect the resource.

Many adaptations that we have implemented focus primarily on tuning attributes of the game (please see Pac-Man and Minecraft examples in Table I), while others focus on content modifications (please see the TileGame example of usage of different versions of maps in Table I).

V. DISCUSSION

In this section, we discuss the benefits of using a software design pattern approach for implementing adaptivity in video games.

A. Reusable Source Code

Reusability refers to the degree to which existing code can be reused in new applications. Since design patterns provide a reusable solution, it is expected that reusable source code can be created for such solutions as well. In [12], we reported an empirical investigation involving source code analysis of the Pac-Man and TileGame games. In that study, we experienced 77.52% and 79.68% code reusability in Pac-Man and TileGame respectively while implementing the adaptive systems using our software design patterns. Recently, we have extended this study to the popular commercial game Minecraft [33] and found comparable results. In Figure 9, we show a summary of these studies, identifying reusable and application-specific logical Source Lines of Code (SLOC). As we can see, 600 SLOC (74.26% in Minecraft; 79.68% in TileGame; and 77.52% in Pac-Man) of the adaptive system remained unchanged across all three games.

Reusability of source code reduces implementation time and increases the probability that prior testing has eliminated defects.

B. Repeatable Process

In our design pattern-based approach, since the high level structure of the solution is already known, it is possible to create a step-by-step method for developing adaptive video games. From our experience in implementing adaptivity into Pac-Man and TileGame [11,12], we formalized such a process and applied it to the Minecraft game. In Table II, we provide a generalized description of the process to incorporate the concepts of adaptive gameplay discussed in the previous section.

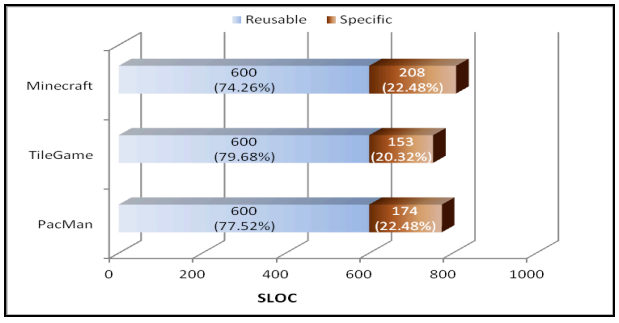


Figure 9. Source code reusability found in adaptive games developed using our design patterns

TABLE II. ADAPTIVE GAME IMPLEMENTATION PROCESS

#	Activity	Output
1	Identify the aspects of the game that will be adaptively adjusted.	
2	For each of the aspects identified in step-1 repeat step-3 to step-9.	
3	Define or reuse available sensors.	Sensors
4	Identify or introduce attributes that can be adjusted.	
5	Identify adaptation scenarios involving sensors and attributes from step-3 and step-4.	
6	Define thresholds based on the scenarios identified in step-5 for the sensors defined in step-3, and define observers to relate thresholds to sensors.	Thresholds, Observers
7	Define triggers to represent each scenario, and develop adaptation detector logic from the scenarios.	Triggers
8	Use attributes identified in step-4 to create decisions to modify game functionality according to the scenarios identified in step-5.	Decisions
9	Define rules to relate triggers to decisions based on the adaptation scenarios identified in step-5.	Rules

A well-defined process for adaptivity is important for industrial adoption as it enables progress tracking, planning, and automation. Furthermore, it allows developers to focus more on gameplay design and adaptive logic design, rather than implementation details. Unlike ad hoc approaches, a well-defined process is repeatable with consistent results across various games. Our study on three different games using the process described above is a primary validation of consistent repeatability of the process. Since the process is defined in a step-by-step method with specific artifacts expected as outputs from each step (please see the third column in Table II), it will be possible to define specific metrics to estimate project size and later measure progress as the project moves forward.

C. Impact on Quality Factors

In [12], we examined how different software quality factors are impacted by the usage of our design patterns. We have already discussed the impact on reusability in subsection A, and so we briefly discuss the impact on other quality factors below.

Integrability: Integrability refers to the ability to make the separately developed components of a system work correctly together. As we can see in Figure 5, the integration points among the design patterns and with the game are clearly defined. Because of these clearly defined integration points, the four design patterns can be integrated with each other and a game rather easily.

Portability: Portability is the ability of a system to run under different computing environments. A framework- or middleware-based approach for creating a self adaptive-system is usually specific to a particular programming language and or platform, whereas a design pattern-based approach is highly portable across different platforms and programming languages [17]. These design patterns were derived from the self-adaptive system literature in the context of adaptivity in video games, with a particular focus on auto dynamic difficulty. This indicates the portability of these design patterns across domains. Also, in our research, we managed to port them (as a solution) from one game to another within the platform (Java). This indicates portability across systems on the same platform. In the future, we plan to examine the portability of these design patterns across platforms as well.

Maintainability: Maintainability refers to the ease of the future maintenance of the system. As discussed earlier, different parts of the design patterns have specific concerns (e.g., *Sensor*s will collect data, *Driver*s will make changes to the game, and so on), and so the resulting source code will have high traceability and maintainability. Furthermore, as the use of these design patterns provides source code reusability (please see Figure 9), this will increase the probability that prior testing has eliminated defects while being used in a new game.

D. Automation

Using our approach, it is possible to implement tools that will guide developers through the process of enabling adaptivity in their games. We are currently designing a semi-automatic tool to help developers to easily integrate a game into the tool and then identify metrics for sensors, brainstorm adaptation scenarios, identify attributes to adjust in the game, maintain traceability between these artifacts, and so on. The benefits of such semi-automatic tools include reducing development effort and defects, standardization, ease of progress tracking, and improving maintainability.

VI. CONCLUDING REMARKS

Adaptivity is becoming increasingly essential to modern video games. Previous attempts at adaptivity in games can be characterized as ad hoc from a software engineering perspective; lacking rigor, structure, and reusability, with custom solutions per game. There is a critical need for software frameworks, patterns, libraries, and tools to enable adaptive systems for games. Thus, in this paper, we leverage the benefits of software design patterns to construct a framework for adaptive games. Based on studies of three different games, including the large commercial game Minecraft, we discussed how the usage of these software design patterns results in a reusable approach both in terms of source code and process and improves a number of other quality aspects.

There are many possible directions for future work in this area. We plan to extend our work, enabling auto dynamic difficulty in additional games, exploring other forms of adaptivity, and bringing our framework to other platforms. While our approach is designed to be generalizable, and work to date supports this, further work is necessary to fully assess this and identify limitations to our approach. To further assess the effectiveness and efficiency of our approach, we will conduct extensive user testing and performance testing. Since a key goal of adaptivity in games is an improved player experience, this user testing is essential. Lastly, to assist developers, we will continue developing semi-automatic and automatic tools to enable adaptivity with minimal effort on their part.

 ISBN: 978-1-61208-274-5

REFERENCES

[1] G. Dolbier and A. Goldschmidt, The Business of Interactive Entertainment. IBM Digital Media Solutions Technical Report G565-1461-00, May 2006.

[2] A. Glassner, Interactive Storytelling: Techniques for 21st Century Fiction. A K Peters, Ltd., 2004.

[3] P. Sweetser, Emergence in Games. Charles River Media, 2008.

[4] G. Andrade, G. Ramalho, H. Santana, and V. Corruble., "Challenge-Sensitive Action Selection: An Application to Game Balancing". In the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Compiègne, France, September 2005, pp. 194-200.

[5] D. Charles, M. McNeill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kücklich, and A. Kerr, "Player-Centred Game Design: Player Modelling and Adaptive Digital Games". Proceedings of DiGRA 2005 Conference: Changing Views Worlds in Play, June 2005, pp. 285-298.

[6] P. Langley, Machine Learning for Adaptive User Interfaces. Kunstiche Intellugenz, 1997.

[7] D. Charles and M. Black, "Dynamic Player Modelling: A Framework for Player-Centered Digital Games". In Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education, Microsoft Campus, 2004, pp. 8-10.

[8] B. Pfeifer, "Creating Emergent Gameplay with Autonomous Agents". Proceedings of the Game AI Workshop at AAAI-04, San Jose, California, July 2004, pp.20.

[9] B. Reynolds. How AI Enables Designers. Appeared in the Proceedings of the 2004 Game Developers Conference, San Jose, California, March 2004, pp. 20.

[10] G.N. Yannakakis and J. Hallam, "Real-time Game Adaptation for Optimizing Player Satisfaction". IEEE Transactions on Computational Intelligence and AI in Games, 1(2), 2009, pp. 121-133.

[11] M. Chowdhury and M. Katchabaw, "Software Design Patterns for Enabling Auto Dynamic Difficulty in Video Games". Proceedings of the 17th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games. Louisville, Kentucky. July, 2012, pp. 76-80.

[12] M. Chowdhury and M. Katchabaw, "Improving Software Quality Through Design Patterns: A Case Study of Adaptive Games and Auto Dynamic Difficulty". Proceedings of GameOn 2012. Magala, Spain. November, 2012, pp. 41-47.

[13] E. Adams, Fundamentals of Game Design, Second Edition. New Riders, 2010.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1995.

[15] M. Csikszentmihalyi., Creativity: Flow and the Psychology of Discovery and Invention. New York, NY: Harper Collins Publishers. 1996.

[16] C. Bailey and M. Katchabaw, "An Experimental Testbed to Enable Auto-Dynamic Difficulty in Modern Video Games". In Proceedings of the 2005 North American Game-On Conference. Montreal, Canada. August 2005, pp. 18-22.

[17] A. Ramirez and B. Cheng, "Design Patterns for Developing Dynamically Adaptive Systems". Proceeding of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. Cape Town, South Africa, May 2010, pp. 49-58.

[18] P. Baillie-de Byl, Programming Believable Characters in Games. Charles River Media, 2004.

[19] J. Dias, S. Mascarenhas, and A. Paiva, "FAtiMA Modular: Towards an Agent Architecture with a Generic Appraisal Framework". Workshop on Standards in Emotion Modeling, Leiden, Netherlands, August 2011, pp. 12.

[20] A. Guye-Vuilleme and D. Thalmann, A High-Level Architecture For Believable Social Agents. Virtual Reality, Volume 5, Number 2, 2001, pp. 95-106.

[21] M. Nelson, C. Ashmore, and M. Mateas, "Authoring an Interactive Narrative with Declarative Optimization Based Drama Management". Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE). Marina del Rey, California, June 2006, pp. 127-129.

[22] P. Spronck, "A Model for Reliable Adaptive Game Intelligence". IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games, 2005, pp. 95-100.

[23] R. Zhao, Applying Agent Modeling to Behaviour Patterns of Characters in Story Based Games. PhD Thesis, University of Alberta, 2010.

[24] K. Compton and M. Mateas, "Procedural Level Design for Platform Games". Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE). Marina del Rey, California, June 2006, pp. 109-111.

[25] C. Pedersen, J. Togelius, and G. Yannakakis, "Optimization of Platform Game Levels for Player Experience". Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), Oct., 2009, pp. 191-192.

[26] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards Automatic Personalised Content Creation in Racing Games". Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games. April 2007, pp. 252-259.

[27] P. Rani, N. Sarkar, and C. Liu. "Maintaining Optimal Challenge in Computer Games Through Real-time Physiological Feedback". Proceedings of the 11th Intl. Conf. on Human-Computer Interaction. Las Vegas, USA, July 2005, pp. 184-192.

[28] R. Hunicke, "The Case for Dynamic Difficulty Adjustment in Games". Proceedings of the 2005 ACM SIGCHI International Conf. on Advances in Computer Entertainment Technology. Valencia, Spain, June 2005, pp. 429-433.

[29] Y. Hao, S. He, J. Wang, X. Liu, J. Yang, and W. Huang., "Dynamic Difficulty Adjustment of Game AI by MCTS for the Game Pac-Man". Proceedings of the Sixth Int. Conference on Natural Computation. Yantai, China, August 2010, pp. 3918-3922.

[30] N. Hocine and A. Gouaïch, "Therapeutic Games' Difficulty Adaptation: An Approach Based on Player's Ability and Motivation". Proceedings of the 16th Intl. Conf. on Computer Games. Louisville, Kentucky, USA, July 2011, pp. 257-261.

[31] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges". In ACM Transactions on Autonomous and Adaptive Systems, Vol. 4, No. 2, Article 14, May 2009, pp. 1-42.

[32] D. Brackeen, B. Barker, and L. Vanhelsuwé, Developing Games in Java. New Riders, 2004.

[33] Mojang, Minecraft. Retrieved from: https://minecraft.net. Last accessed: Jan 29, 2013.

[34] MCP Team, Main Page – Minecraft Coder Pack, Retrieved from: http://mcp.ocean-labs.de/. Last accessed: Jan 29, 2013.

 ISBN: 978-1-61208-274-5

A Gravitational Approach for Enhancing Cluster Visualization in Self-Organizing Maps

Leonardo Enzo Brito da Silva, José Alfredo Ferreira Costa
Departamento de Engenharia Elétrica
Universidade Federal do Rio Grande do Norte
Natal, Brazil
{leonardoenzob, jafcosta}@gmail.com

***Abstract*—This paper presents a modified gravitational clustering algorithm applied to the neurons of self-organizing maps in order to enhance the visualization of clusters through the U-matrix technique. For a given neuron, the proposed method considers the attraction among its k nearest neighbors in the data space, where k decreases monotonically over time and its value may vary according to the local pattern density. The attraction between neurons that are considered as not belonging to the same close-knit group is penalized. The results obtained for some synthetic and real world data sets are presented.**

Keywords-self-organizing maps; gravitational clustering; visualization techniques

I. INTRODUCTION

Nowadays, a plethora of data from the most diverse sources is collected and stored [1]. Data mining is one of the fields that aim to transform this information into useful knowledge. Among the main problems faced in this field, may be cited the data scalability and dimensionality, as well as its complexity, heterogeneity and quality (noise and outliers). Therefore, the analysis of databases requires careful interpretation of the results obtained with the mathematical models and the visualization techniques applied [2]. Visualization consists of the conversion of the data attributes into a visual structure, so as to observe its characteristics and properties [3].

The self-organizing maps (SOM) [4] are artificial neural networks widely used in the data mining field, mainly due to the mapping of a high dimensional input space (data space) to an output space of lower dimensionality (fixed grid of neurons), while preserving data topology. In this sense, the SOM network is a nonlinear generalization of principal component analysis [5]. It is used for clustering as well as for visualization. The U-matrix [6] is a well-known visualization technique associated with the SOM network. Its main issue concerns its resolution when applied to decreasing map sizes, i.e. on small maps the visualization is compromised, while on large maps the definition of clusters becomes increasingly clear in data sets where distance metrics are relevant.

Recently, gravitational-based clustering algorithms have been used to perform the clustering task [7]-[9]. These algorithms are hierarchical and agglomerative, i.e. they progressively define clusters from a database given a similarity metric, while forming a tree structure: in its base, each pattern is a cluster, and, at the top, there is only one cluster.

This paper focuses on improving the U-matrix visualization through the application of an algorithm based on the gravitational principles on the SOM neurons, as a way of increasing inter-cluster distances and decreasing intra-cluster distances. More specifically, the objective is to determine an updating rule for the weights associated with the SOM neurons, in order to perform a post-processing and provide a visualization in which separation between clusters is sharper.

The remainder of the paper is organized as follows. Section II provides general considerations of the SOM network, while Section III discusses some of its well-known visualization techniques. In Section IV, a brief description of some gravitational algorithms is provided. In Section V, the proposed method is defined, and, in Section VI, the data sets used in the experiments are concisely described. The simulation results and discussions are presented in Section VII. In Section VIII, some conclusions are drawn.

II. SELF-ORGANIZING MAPS

Self-organizing maps consist of a set of topologically ordered neurons situated in a static lattice (output space). The neuron grid can be either rectangular or hexagonal, differing in the number of immediate neighbors - four or six respectively. In general, the network grids are 1-D or 2-D. Although higher dimensionalities are possible, they generally are not used, since visualization becomes more difficult or not even feasible. Each neuron has an associated weight vector in the data space (input space), so a projection from a higher to a lower dimensional space is obtained. The SOM network can be seen as an adaptive vector quantization algorithm. The learning process involved is unsupervised and encloses the following three principles: competition, cooperation and adaptation. For each pattern presented to the network, the neurons compete with each other, so that a winner (best matching unit - BMU) is defined as the one with the minimum Euclidean distance to this pattern:

$$\|\mathbf{x}_i - \boldsymbol{w}_i\| < \|\mathbf{x}_i - \boldsymbol{w}_l\| \quad \forall l \neq i \tag{1}$$

where $\|\cdot\|$ is the Euclidean norm, $\boldsymbol{w} \in \mathfrak{R}^{\boldsymbol{d}}$ is a weight

 ISBN: 978-1-61208-274-5

vector ($\boldsymbol{w}_i$ is the BMU), $\mathbf{x}_i \in \mathfrak{R}^d$ is a pattern from the data set, and d is the dimension of the data space. However, not only the winner neuron, but its neighborhood also participates in the learning process. The adaptation rule is given by (2) [4]

$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + \eta(t) h_{i,j}(t)\left[\mathbf{x}_i - \boldsymbol{w}_j(t)\right] \quad (2)$$

where t is time, $\boldsymbol{w}_j(t)$ is the weight vector associated with the jth neuron, $\mathbf{x}_i$ is the ith pattern from the data set presented to the network, $\eta(t)$ is the learning rate, $h_{i,j}(t)$ is the neighborhood kernel. The neighborhood kernel (which is centered on the BMU and is usually Gaussian) and the learning rate must be monotonically decreasing as the training algorithm progresses [4].

During the training stage, the SOM network behaves as an elastic net that molds itself to the intrinsic shape formed by the patterns. The neuron's placement reflects the data set density distribution: the number of neurons in a certain region of the input space is related to the number of patterns in that region, what is known as the magnification factor.

The quality of a given trained SOM can be measured by the following figures of merit: the quantization error (3) [4] and the topographic error (4) [10],

$$q_e = \frac{1}{N}\sum_{k=1}^{N}\left\|\boldsymbol{x}_k - \boldsymbol{w}_{BMU}^{x_k}\right\| \quad (3)$$

$$t_e = \frac{1}{N}\sum_{k=1}^{N} f(\boldsymbol{x}_k) \quad (4)$$

where N is the number of patterns of the data set, $\mathbf{x}_k$ is the kth pattern from the data set, $\boldsymbol{w}_{BMU}^{x_k}$ is the BMU of the pattern $\mathbf{x}_k$. The function $f(\boldsymbol{x}_k)$ is equal to zero if the first and second BMU of the pattern $\boldsymbol{x}_k$ are adjacent. Otherwise the function $f(\boldsymbol{x}_k)$ is equal to one.

The quantization error discloses the network resolution, while the topographic error depicts when there is a divergence between the neighborhood of neurons in the input and output spaces. An extensive discussion of topology in neural networks based on vector encoding can be found in [11].

III. VISUALIZATION TECHNIQUES

In order to suitably view clusters in a given trained SOM network, visualization techniques must be applied. That is, a post-processing stage using its prototypes is needed so as to infer characteristics of the dataset. Typically, visualization techniques take into account not more than one metric within its definition, for example, distances between prototypes, as the U-matrix, component planes [12], component gradients matrix [13], or pattern density, as in the hit histogram, P-matrix [14], CONNvis [15] and Smoothed Data Histogram (SDH) [16]. There are methods that take into consideration both distance among prototypes and pattern density associated with them, such as the CONNDISTvis [17] and the U*-matrix [18].

The U-matrix is one of the most popular visualization techniques, and consists of a matrix whose positions are filled with the Euclidean distances between the neurons in the data space. Consider that a map has a rectangular grid of size $a \times b$, then the U-matrix has the size $(2a-1) \times (2b-1)$. The relative positions of the neurons themselves in the matrix are obtained by a function f of the neighboring distances in the grid, where generally f is a mean or a median function of neighboring values. The Euclidean distances in the U-matrix can be calculated on the basis of all attributes or specific ones with the use of masks. The particular case where a U-matrix is calculated for each attribute of data form the component planes.

The P-matrix aims to estimate the probability density of the data [19]-[20]. It has a structure that is the same size as the map grid. In the P-matrix, the value at the position related to the neuron $\boldsymbol{w}_i$ consists of the number of patterns inside a hypersphere of radius r centered on that neuron. The radius is a fixed parameter for all neurons and is called Pareto radius. The U*-matrix is an enhanced visualization method that is generated by using information provided by both the U-matrix and the P-matrix. It has the same size of the latter. Its value U^* for the relative position of each neuron $\boldsymbol{w}_i$ in the grid is obtained by (5) [18]

$$U^*(\boldsymbol{w}_i) = U(\boldsymbol{w}_i)\left[\frac{P(\boldsymbol{w}_i) - \bar{P}}{\bar{P} - P_{max}} + 1\right] \quad (5)$$

where $P(\boldsymbol{w}_i)$ and $U(\boldsymbol{w}_i)$ are the values of the P-matrix and U-matrix associated with the neuron $\boldsymbol{w}_i$, $\bar{P}$ and P_{max} are the mean and maximum values of the P-matrix, respectively.

IV. GRAVITATIONAL CLUSTERING ALGORITHMS

The gravitational algorithm and its application to the clustering task were first proposed by Wright [21] and rely on the law of universal gravitation. It may be classified as an agglomerative hierarchical algorithm, as it begins with a set of N objects and ends with only one. However, contrary to classic hierarchical agglomerative algorithms where patterns are static, in the gravitational algorithm all patterns are considered as mobile particles subjected to the gravitational fields of one another. The results obtained using such method can be seen as a dendrogram. The strength of a given cluster structure is greater the larger the interval of time during which the system remains in that clustering state.

A variant of the clustering algorithm was proposed by Gomez et al. [7], in order to automatically determine the number of clusters, remove noise and generate prototypes representing the database. This approach differs from the latter in the sense that the particles are always considered with unitary mass (as opposed to the original algorithm where the mass changes when there is a mergence), and the stopping criterion is the number of iteration. The algorithm is very sensitive to the gravitational constant and its decay function, which may lead to a generation of only one cluster or none at all. According to the model, the particles move as described in (6)-(7) [7]

 ISBN: 978-1-61208-274-5

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \left(\frac{G}{\left\| \boldsymbol{d}\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) \right\|^3} \right) \boldsymbol{d}\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) \quad (6)$$

$$\boldsymbol{d}\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) = \quad \boldsymbol{y}(t) - \boldsymbol{x}(t) \quad (7)$$

where G is the gravitational constant that decreases monotonically over time, $\boldsymbol{x}$ and $\boldsymbol{y}$ are patterns randomly chosen from the dataset, and $\|\cdot\|$ is the Euclidean norm. Particles are merged when separated by a minimum distance which is an input parameter.

It is also stated that a good performance of the algorithm does not necessarily need the entire database to be used, which thereby opens the possibility to use vector quantization techniques, such as the SOM network, before its application. Therefore, recently, a gravitational clustering of the SOM (gSOM) [8] based on the work of Gomez et al. [7] was proposed. It consists of two steps: in the first phase, a SOM network is trained with the data set. In the second phase, the interpolating neurons, that is, those neurons that are not associated with any pattern are eliminated, as well as their connections. The gravitational algorithm is then applied to the remaining neurons.

Each time a random pair of objects is selected, according to predefined probability functions, the gravitational algorithm is applied. The stopping criterion can be either the number of iterations or the maximum number of clusters to be found. The gSOM has also been used in a clustering ensemble [9], in which the different partitions obtained, due to its stochastic nature, are analyzed in a consensus function in order to define the final partition. Other variants of the gravitational algorithms and applications were proposed in the literature, such as [22]-[23].

V. PROPOSED APPROACH

The proposed method, the k-gSOM algorithm, is concerned with distance information between close neurons on the map as well as pattern density in their vicinity. It is assumed that due to the attraction exerted by the patterns to the neurons in the network, the overwhelming majority of neurons are located in high density places, while a minority make the connection between these groups (interpolating neurons). The proposed method relates to the work of Ilc and Dobnikar [8], consisting of two stages: in the first stage, a SOM network is trained using its standard algorithm, and, in the second stage, the proposed method is applied to the network neurons.

The technique is based on gravitational principle and on a hit histogram variant. It is a gravitational clustering algorithm as the neurons are subjected to attraction forces of one another, and they all tend to gather at the same position when time becomes sufficiently large. The information of pattern density and distance among close neurons is used to adapt their weights and collapse them in order to obtain an improved U-matrix visualization.

The hit histogram consists of an accumulation array of the same size as the map, where each bin is associated with the position of a neuron in the SOM grid. The hit histogram depicts the number of patterns that each neuron is the BMU. As opposed to the U-matrix, the information provided by a hit histogram is more useful when dealing with small sized maps, in which the pattern to neuron ratio is usually greater than 1. Otherwise, due to the dissolution phenomenon, the matrix associated with the hit histogram becomes very sparse, which impedes the proper display of the data characteristics. The P-matrix and SDH are examples of visualization techniques that surpass this issue by using hyperspheres with Pareto radius and considering more than one BMU for each pattern, respectively. In this work, the values of the hit histogram consist of how many patterns are within a hypersphere centered on each neuron. The radius of the hypersphere is regarded as the minimum for which all the neurons have at least one associated pattern. By doing this the division by zero in (9) is avoided (the neuron masses are associated with the hit histogram) as there are no neurons without a pattern associated.

Therefore, a neuron $\boldsymbol{w_j}$ at time t will be moved according to the attraction forces among its k_j neighbors $\boldsymbol{w_i}$. The pairwise force between neurons $\boldsymbol{w_j}$ and $\boldsymbol{w_i}$ is related to their proximity in the input space and the ratio of patterns shared by their associated hyperspheres. The overall number of neighbors k_j depends on whether $\boldsymbol{w_j}$ is in a denser region or not. The direction and magnitude of the movement is given by the resultant of all the attraction forces between $\boldsymbol{w_j}$ and its neighbors. The movement will occur until the stopping criterion is reached, which is the number of iterations. The equations governing the adaptation are as follows

$$\boldsymbol{w_j}(t+1) = \boldsymbol{w_j}(t) + \Delta \boldsymbol{w_j}(t) \quad (8)$$

$$\Delta \boldsymbol{w_j}(t) = \frac{1}{m_j(t) k_j(t)} \sum_{i=1}^{k_j(t)} \left\{ \frac{\big[1 + JC_{i,j}(t)\big]\big[1 - d_{i,j}(t)\big]}{p_{i,j}(t)} \times \big[\boldsymbol{w_i}(t) - \boldsymbol{w_j}(t)\big] \right\} \quad (9)$$

$$k_j(t) = k_{max}(t) H_j(t) \quad (10)$$

$$d_{i,j}(t) = \left\| \boldsymbol{w_i}(t) - \boldsymbol{w_j}(t) \right\| \quad (11)$$

where $\boldsymbol{w_j}$ is the jth neuron, k_j is the effective number of neighbors of neuron $\boldsymbol{w_j}$. The parameter k_j is proportional to the pattern density where $\boldsymbol{w_j}$ is located, and its maximum possible value is predetermined at time t as k_{max}. The $\|\cdot\|$ is the Euclidean norm, m_j is the mass of the neuron $\boldsymbol{w_j}$ and corresponds to the number of patterns inside the hypersphere centered on $\boldsymbol{w_j}$ at time t. The distance $d_{i,j}$ is normalized in the interval $[0;1]$ regarding all pairwise distances between neurons, and also negated in (9) so as to be transformed from a dissimilarity to a similarity measure.

The parameter $JC_{i,j}$ is the Jaccard coefficient [24] defined by the ratio of the intersection and union cardinalities of the sets containing the patterns covered by the hyperspheres of the neurons $\boldsymbol{w_j}$ and $\boldsymbol{w_i}$ at time t

$$JC_{i,j}(t) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} \quad (12)$$

where $|\cdot|$ is the set cardinality, S_i and S_j are the number of patterns inside the hyperspheres centered on the neurons $\boldsymbol{w_i}$ and $\boldsymbol{w_j}$, respectively. The parameter $p_{i,j}$ is defined as

$$p_{i,j}(t) = \begin{cases} \frac{1}{n}\sum_{q=1}^{n} m_q, & if\ d_{i,j}(t) > \alpha \\ 1, & otherwise \end{cases} \quad (13)$$

where m_q corresponds to the mass of one of the n neurons whose distance to $\boldsymbol{w_j}$ is less or equal to the parameter α at time t (Fig. 1). The attraction force is penalized by the parameter $p_{i,j}$ if a neuron $\boldsymbol{w_i}$ is very far regarding a close group of neurons around $\boldsymbol{w_j}$ that is defined by α. Thus, a neuron $\boldsymbol{w_i}$ cannot attract a single neuron from within this group, but in fact, the whole group, thereby diminishing the attraction force and compensating k_j if it is overly estimated.

The attraction among a decreasing number of neighboring neurons in the input space is considered as the algorithm progresses. For each neuron $\boldsymbol{w_j}$, at each iteration, the effective number of neighbors is a fraction of k_{max} that is proportional to the density of patterns in the region that the neuron is currently situated: H_j is the value of the hit histogram generated at time t and associated with neuron $\boldsymbol{w_j}$. The values of H_j are normalized in the range [0.1;1] so k_j is a nonzero percentage of k_{max}. By doing this we prevent that neurons in small clusters have the same neighborhood size as neurons in large clusters, and therefore reducing the influence of the latter over the first. The role of the parameter α consists of defining the minimum distance for which a set of neurons should be considered as a group. It relates to the minimum distance of mergence in the traditional gravitational algorithms. However, in the proposed method neurons are not merged nor eliminated: the number of neurons is constant throughout the algorithm steps, there is only an update to their position in the input space.

The Jaccard coefficient is included so as to add a second term of attraction between neuron $\boldsymbol{w_j}$ and a given neighbor $\boldsymbol{w_i}$: if they have patterns in common while considering a given hypersphere, they should be brought together proportionally to the intersection divided by the overall patterns associated with them. At each iteration of the algorithm, the pairwise distances between all prototypes are calculated, as well as the hypersphere radius, the neuron masses, the Jaccard coefficients and the effective number of neighbors for each neuron, before using their respective values in (9). The parameters $JC_{i,j}$, m_j and k_j are ultimately dependent on the radius of the hypersphere, which is calculated at each iteration of the algorithm. The attraction among a decreasing number of neighboring neurons in the input space is considered since k_{max} is monotonically

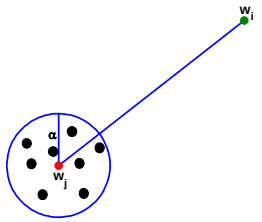


Figure 1. Illustrative case where the attraction between $\boldsymbol{w_j}$ (red dot) and $\boldsymbol{w_i}$ (green dot) is penalized by $p_{i,j}$ (mean mass of the group of neurons inside the circle of radius α). All neurons whose distances to $\boldsymbol{w_j}$ are less or equal to α are considered as belonging to the same close-knit group (black dots), and therefore the parameter $p_{i,j}$ related to their attraction forces is equal to unity.

decreasing while the algorithm progresses, as well as the parameter α. In this work, both k_{max} and α were set to decrease linearly with time t according to (14)-(15)

$$k_{max}(t) = (k_0 - k_f)\left(1 - \frac{t}{T}\right) + k_f \quad (14)$$

$$\alpha(t) = (\alpha_0 - \alpha_f)\left(1 - \frac{t}{T}\right) + \alpha_f \quad (15)$$

where T is the total number of iterations, α_0 and α_f are the initial and final values of α, respectively. The parameters k_0 and k_f are the initial and final values of k_{max}, respectively.

The summary of the algorithm is presented in Table I:

TABLE I. K-GSOM ALGORITHM

1. Initialize k_{max} and α as well as their decreasing functions.
2. Determine the hypersphere radius, calculate the masses of each neuron and generate the normalized hit histogram H.
3. Calculate and normalize the pairwise distance between all neurons.
4. For each prototype $\mathbf{w_j}$
a. Find k_j nearest neighbors by multiplying the current k_{max} by the value of H_j in the position associated to $\mathbf{w_j}$.
b. Calculate $\mathbf{\Delta w_j}$. If a neuron $\mathbf{w_i}$ is not close enough to $\mathbf{w_j}$ (distance defined by the parameter α) their attraction is penalized by dividing it by $p_{i,j}$, otherwise $p_{i,j}$ is equal to unity.
5. Update $\mathbf{w_j}$ (sequential algorithm).
6. If the stopping criterion was not met, return to step 2.

VI. DATA SETS

The proposed method was applied to the following synthetic data sets from the Fundamental Clustering Problem Suite [25]: *Hepta* and *Tetra*. Another artificial dataset consisting of two Gaussian clusters mixed with noise was considered. The *Wine* data set [26] was also used in the experiments. All datasets were normalized in the hypercube $[0;1]^n$ as a pre-processing stage. The *Tetra* data set consists of 400 patterns that form four very close clusters in $\Re^3$ so that density information is more relevant than distance among prototypes. The *Hepta* data set consists of 212 patterns that form seven well defined clusters in $\Re^3$, each

one with different variances. The *Noisy Gaussian* data set consists of two Gaussian clusters with 400 patterns and 100 patterns that represent noise. The *Wine* data set is a real world database that consists of 178 patterns that forms three clusters in a 13 dimensional space. All previously mentioned data sets are depicted in Fig. 2.

VII. RESULTS AND DISCUSSION

The experiments were carried out with SOM networks whose grid sizes were all 10x10, and were trained in the first stage using the SOM Toolbox [27]. For the second stage, the initial value of k_{max} was set to 80% of the total number of neurons and it was decreased linearly over the iterations until it reaches 1. The parameter α was also decreased linearly over the iterations from 10^{-1} to 10^{-3}.

The SOM network trained with the *Noisy Gaussian* dataset is depicted in Fig. 3, along with the distances to the closest pattern to each neuron. The maximum pairwise distance between a neuron and its closest pattern is then used as the hypersphere radius; therefore the least populated hypersphere will have 1 pattern. This information is used in order to generate the hit histogram and to calculate the Jaccard coefficient (Fig. 4). The steps of the algorithm shown in Figs. 3 and 4 are repeated continuously. The movement of neurons as the proposed algorithm progresses is depicted in Fig. 5. After 250 iterations, the final placement of the neurons is depicted in Fig. 6, as well as the values over time for: the hypersphere radius, the parameters k_{max} and α. The effective neighbor number for each neuron at each iteration is shown in Fig. 7.

It is perceptible in Fig. 6(d) that the radius tends to a permanent regime, that is, after a certain number of iterations it remains in a specific value with small fluctuations. Therefore, the computational cost may be reduced by setting the stop criterion as the sum of the radii differences between one iteration to the next: if it remains within a certain range ε for a fixed number of iterations then the algorithm stops. In Fig. 8, the trained SOM network and the best results of the proposed algorithm are shown. They were obtained for the *Tetra*, *Hepta*, *Noisy Gaussian*, and *Wine* data sets after 85, 125, 250 and 190 iterations, respectively. The Fig. 9 (a) and (b) depict the U-matrix and the U*-matrix (generated using the SOMVIS Package) obtained from the original SOM, respectively. In Fig. 9 (c) the U-matrix of the SOM network resulting from the application of the proposed method is shown.

As depicted in Fig. 9, the borders of the clusters are visually sharper than the original U-matrix and the U*-matrix. Albeit the final positions of the neurons do not correspond to the positions of the clusters' centroids (phenomenon resulting from the gravitational effect) a repositioning may be achieved by relating the neurons that converged to a centroid to their original map positions in the input space.

In order to measure the performance of the method, first the MBSAS [28] was applied so as to obtain the centroids resulting from the neurons' movement. The radius parameter was set to α_f. The number of centroids found and the prototypes they represent were stored and separated into

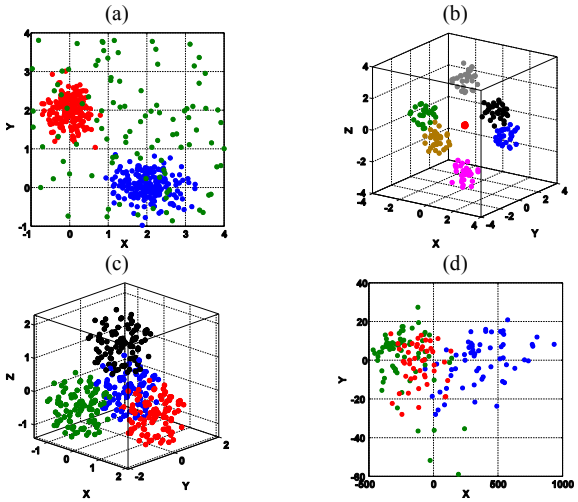


Figure 2. (a) Elements of the *Noisy Gaussian* data set. (b) Elements of the *Hepta* data set. (c) Elements of the *Tetra* data set. (d) Elements of the *Wine* data set using a 2-D PCA (principal component analysis) projection. Each class in each data set is depicted in a specific color.

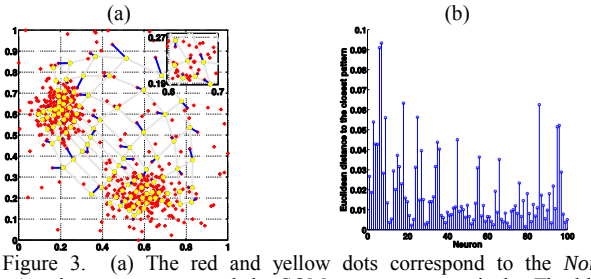


Figure 3. (a) The red and yellow dots correspond to the *Noisy Gaussian* data set patterns and the SOM neurons, respectively. The blue lines indicate which is the closest pattern to each neuron. (b) Stem plot regarding the Euclidean distances of the closest pattern to each neuron.

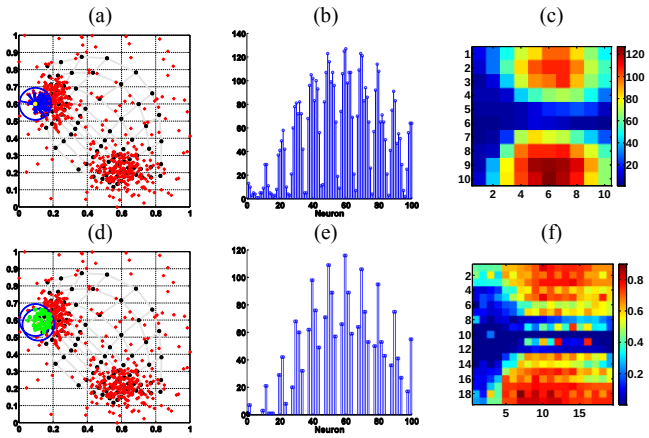


Figure 4. (a) The red and black dots correspond to the data set patterns and neurons, respectively. The yellow dot is a neuron which is the center of its correspondent hypersphere, which is depicted as the blue circle. All patterns inside this circle are linked to the neuron by blue lines. (b) Stem plot of the number of pattern inside each neuron hypersphere. (c) Hit histogram generated with the number of patterns inside the hyperspheres. (d) The red and black dots correspond to the data set patterns and neurons, respectively. The green dots are the intersection between the circles around two neighboring neurons. (b) Stem plot of the number of patterns in the intersection of neurons $\boldsymbol{w_i}$ and $\boldsymbol{w_j}$. (f) Jaccard coefficient matrix whose values are calculated among neurons that are 4-neighbor on the lattice (output space).

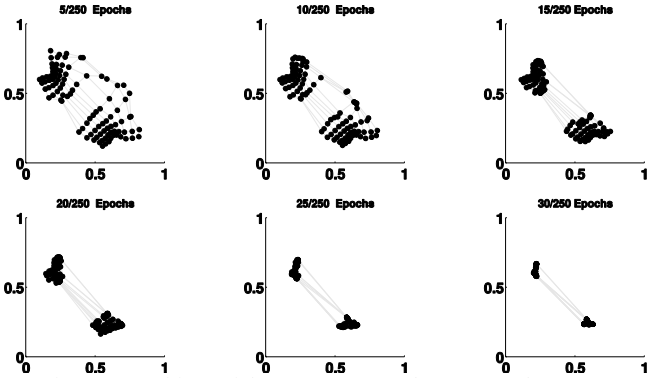


Figure 5. Positions of the neurons at epochs 5 to 30. The neurons are gathering in the densest regions of the *Noisy Gaussian* data set.

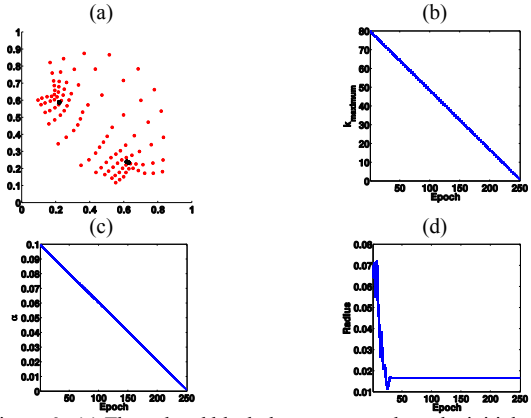


Figure 6. (a) The red and black dots corresponds to the initial and final positions of the neurons, respectively. The evolution of each parameter over time is shown in: (b) maximum possible number of neighbors (c) maximum distance for which neurons are considered as belonging to the same group (d) hypersphere radius for a given neuron $\boldsymbol{w_j}$.

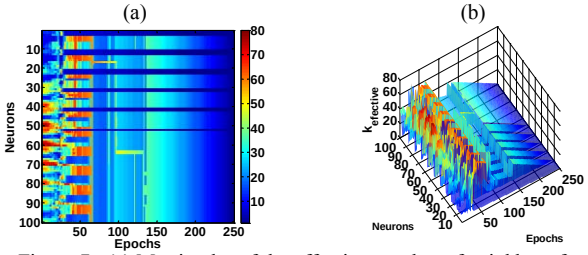


Figure 7. (a) Matrix plot of the effective number of neighbors for each neuron over time (b) Surface plot of 'a'. Neurons in regions with fewer patterns are seen as streaks or valleys, as it is expected. The effective number of neighbors is a fraction of k_{max} that is proportional to the density of the region the neuron is located.

classes, that is, which neurons converged to each position (see the representative colors depicted in Fig. 8). Then a cross tabulation was performed with the datasets' man given groundtruth so as to appropriately compare the classes. The classification accuracies were then calculated (see Table II) in order to evaluate the partitions visible in the new U-matrix. The classification accuracy is defined as (16) [29]

$$Clas\ acc = \frac{\#\ correct\ classified\ patterns}{\#\ patterns} \qquad (16)$$

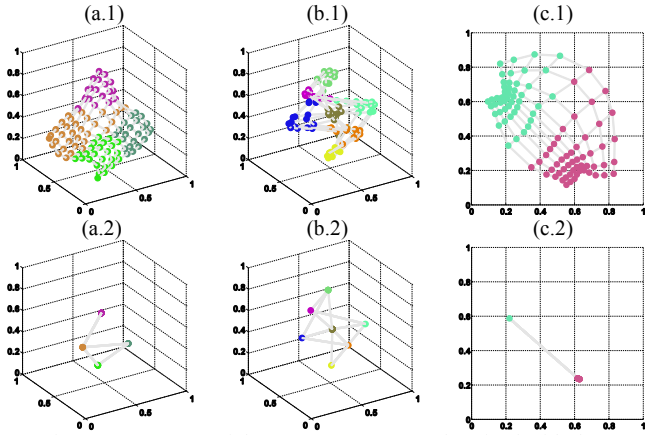


Figure 8. Neurons of the 10x10 SOM network trained with the *Tetra* (a.1), *Hepta* (b1) and *Noisy Gaussian* (a.1) data sets. Neurons resulting from the application of the proposed method in 'a.1', 'b.1' and 'c.1' are depicted in (a.2), (b.2) and (c.2), respectively. Neurons with the same color in the plots with indexes '1' converged to the same point in their associated plots with indexes '2'.

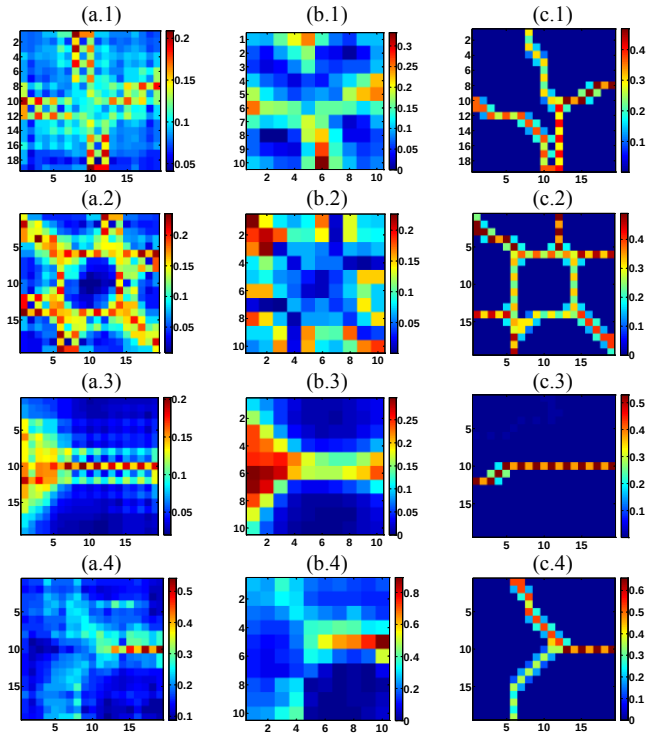


Figure 9. U-matrix (a) and U*-matrix (b) of the 10x10 trained self-organizing maps. U-matrix (c) of the SOM network whose neurons result from the application of the proposed method. The indexes 1 to 4 correspond to the following data sets: *Tetra* (1), *Hepta* (2), *Noisy Gaussian* (3) and *Wine* (4).

TABLE II. PERFORMANCE SUMMARY

Data set	*Number of centroids found*	*Classification accuracy*
Tetra	4	0.9775
Hepta	7	1
Noisy Gaussian	2	1
Wine	3	0.9719

 ISBN: 978-1-61208-274-5

The proposed algorithm is dependent of the iteration number, since the quantity of neighbors was defined as a function of it. In the experiments no universal value for the iteration number was suitable for all datasets as it affects the parameters k_{max} and α, and therefore, it must be set for each database.

VIII. CONCLUSIONS AND FUTURE WORK

A gravitational approach for enhancing the cluster visualization through the U-matrix technique was presented. It takes advantage of the U-matrix increasingly higher resolution while using larger SOM grid sizes and the neurons concentration achieved with the gravitational algorithm. The main concern is to define the neighborhood size and number of iterations as they directly influence the quality of the final map. The experiment parameters were kept equal for all data sets, except for the number of iterations. As the latter becomes larger, the neurons tend to converge to the same point, as it is expected from a gravitational algorithm. In all cases, the proposed approach was able to provide an improved visualization of the clusters using the U-matrix that was generated with the repositioned neurons.

The final result of the proposed algorithm does not represent the real position of the cluster centers due to the tendency of all particles to collapse at the same point, however as the interest consists primarily in visualizing and defining the number of clusters, it is not considered an issue, as the geometry relations are preserved and the visualizations obtained are sharper while remaining coherent.

Future works will focus on an heuristic for automatic selection of the total number of iterations, the parameters k_{max} and α, as well as their decreasing functions based on information from the data and SOM network, in order to achieve a suitable combination regarding the tradeoff between results and the computational cost.

REFERENCES

[1] D. T. Larose, Discovering Knowledge In Data, John Wiley & Sons, 2005.

[2] R. Xu and D. C. Wunsch II, "Survey of clustering algorithms," IEEE Transactions on Neural Networks, vol. 16, no. 3, May 2005, pp. 645–678.

[3] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. Addison Wesley, 2006.

[4] T. Kohonen, Self-Organizing Maps, 3rd ed., Springer-Verlag, 2001.

[5] H. Ritter, "Self-organizing feature maps: Kohonen maps," The Handbook of Brain Theory and Neural Networks, 1995, pp. 846–851.

[6] A. Ultsch and H. P. Siemon, "Kohonen's self organizing feature maps for exploratory data analysis," Proc. of the International Neural Networks Conference (INNC 90), 1990, pp. 305–308.

[7] J. Gomez, D. Dasgupta, and O. Nasraoui, "A New Gravitational Clustering Algorithm," Proc. of the third SIAM International Conference on Data Mining, 2003.

[8] N. Ilc and A. Dobnikar, "Gravitational clustering of the self-organizing map," Adaptive and Neural Computing Algorithms, Lecture Notes in Computer Science, vol. 6594, 2011, pp 11–20.

[9] N. Ilc and A. Dobnikar, "Generation of a clustering ensemble based on a gravitational self-organising map," Neurocomputing, vol. 96, Nov. 2012, pp. 47–56.

[10] K. Kiviluoto, "Topology Preservation in Self-Organizing Maps", Proc. of the IEEE International Conference on Neural Networks, vol. 1, 1996, pp. 294–299.

[11] H.-U. Bauer, J. M. Herrmann, and T. Villmann, "Neural maps and topographic vector quantization," Neural Networks, vol. 12, no. 4–5, Jun. 1999, pp. 659–676.

[12] J. Vesanto, "SOM-based data visualization methods," Intelligent Data Analysis, vol. 3, no. 2, Aug. 1999, pp. 111–126.

[13] J. A. F. Costa, "Uma nova abordagem para visualização e detecção de agrupamentos em mapas de Kohonen baseado em gradientes das componentes," Learning and NonLinear Models, vol. 9, no. 1, 2011, pp. 20–31.

[14] A. Ultsch, "Maps for the visualization of high-dimensional data spaces," Proc. of the Workshop on Self-Organizing Maps (WSOM 03), 2003, pp. 225–230.

[15] K. Taşdemir and E. Merényi, "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps," IEEE Transactions on Neural Networks, vol. 20, no. 4, Apr. 2009, pp. 549–562.

[16] E. Pampalk, A. Rauber, and D. Merkl, "Using Smoothed Data Histograms for Cluster Visualization in Self-Organizing Maps," Proc. of the International Conference on Artificial Neural Networks (ICANN 02), 2002, pp 871–876.

[17] K. Taşdemir, "Graph Based Representations of Density Distribution and Distances for Self-Organizing Maps," IEEE Transactions on Neural Networks, vol. 21, no. 3, Mar. 2010, pp. 520–526.

[18] A. Ultsch, "U*-Matrix : a Tool to visualize Clusters in high dimensional Data," Technical Report no. 36, Dept. of Mathematics and Computer Science, University of Marburg, Germany, 2003.

[19] A. Ultsch, "Proof of Pareto's 80/20 Law and Precise Limits for ABC-Analysis," Technical Report no. 02 / c, University of Marburg, Germany, 2002.

[20] A. Ultsch, "Pareto Density Estimation: A Density Estimation for Knowledge Discovery," Proc. of the 27th Annual Conference of the German Classification Siciety (GfKl 03), 2003, pp. 91–100.

[21] W. E. Wright, "Gravitational clustering," Pattern Recognition, vol. 9, no. 3, Oct. 1977, pp. 151–166.

[22] T. Long and L.-W. Jin, "A New Simplified Gravitational Clustering Method for Multi-prototype Learning Based on Minimum Classification Error Training," Advances in Machine Vision, Image Processing, and Pattern Analysis, Lecture Notes in Computer Science, vol. 4153, 2006, pp. 168–175.

[23] J. Lange and H. Freiesleben, "A parameter-free non-growing self-organizing map based upon gravitational principles: Algorithm and applications," Artificial Neural Networks — ICANN 96, Lecture Notes in Computer Science, vol. 1112, 1996, pp. 827–832.

[24] A. K. Jain and R. C. Dubes, Algorithms for Clustering Data, Prentice Hall, 1988.

[25] A. Ultsch, "Clustering with SOM: U*C," Proc. of the Workshop on Self-Organizing Maps (WSOM 05), 2005, pp. 75–82.

[26] A. Frank and A. Asuncion, "UCI Machine Learning Repository," 2010.

[27] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, "Self-Organizing Map in Matlab: the SOM Toolbox," Proc. of the Matlab DSP Conference, 2000, pp. 35–40.

[28] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th ed., Academic Press, 2008.

[29] M. Meila and D. Heckerman, "An experimental comparison of model-based clustering methods," Machine Learning, vol. 42, no. 1–2, 2001, pp. 9–29.

 ISBN: 978-1-61208-274-5

Model-driven Self-optimization Using Integer Linear Programming and Pseudo-Boolean Optimization

Sebastian Götz, Claas Wilke, Sebastian Richly, Christian Piechnick, Georg Püschel and Uwe Aßmann
Technische Universität Dresden, Software Engineering Group
Dresden, Germany
Email: {sebastian.goetz1,claas.wilke,sebastian.richly,christian.piechnick,goerg.pueschel,uwe.assmann}@tu-dresden.de

***Abstract*—The development of self-optimizing software systems usually requires developers to apply optimization techniques manually, which is time consuming and prone to error. The application of model-driven software development combined with models at runtime takes this burden from developers by generating optimization problems using model transformations. In this paper, we present two such approaches applying integer linear programming and pseudo-boolean optimization. Furthermore, we provide a scalability analysis of both approaches showing their feasibility for pipe-and-filter applications.**

Keywords*—*self-adaptive systems; integer linear programming; pseudo-boolean optimization; MDSD

I. Introduction

The future of software systems is predicted to be characterized by ubiquitous, interconnected software components, running on several heterogenous resources that are subject to frequent changes and optimize themselves w.r.t. their non-functional behavior [1], [2].

In this paper, we address a particular problem of such self-optimizing software systems: the burden of developers to apply optimization techniques manually, which is time consuming and prone to error. We propose to generate optimization problems from models, being more natural to the developers. Thus, we propose the application of model-driven software development, especially model transformations, and the models at runtime paradigm [3] to develop self-optimizing software systems.

We present two approaches: an Integer Linear Programming (ILP)-based and a Pseudo-Boolean Optimization (PBO) [4]-based solution. Both techniques belong to combinatorial optimization [5]. They are suited, because the system configurations, amongst which the best is searched, are combinations of decisions (e.g., which implementation to choose). We compare both approaches and evaluate them w.r.t. scalability, showing their applicability despite their high complexity.

The core contributions of this paper are:

- A runtime optimization approach using ILP.
- A runtime optimization approach using PBO.
- A scalability analysis of both approaches.

The remainder of this paper is structured as follows. In Sect. II a model-driven architecture for self-optimizing software systems is presented being the basis for both the ILP-based solution discussed in in Sect. III and the PBO-based solution discussed in Sect. IV. The scalability of both approaches is examined in Sect. V. Finally, in Sect. VI we outline related work and conclude the paper in Sect. VII.

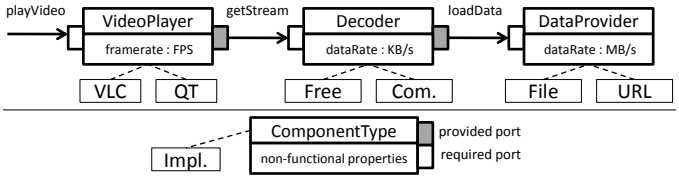


Fig. 1. VideoPlayer SW components and implementations.

II. A Contract-based Architecture for Self-Optimizing Systems

To design and model self-optimizing systems, we developed the non-functional property (NFP)-aware Cool Component Model (CCM) [6] and the Quality Contract Language (QCL) [7]. The CCM provides concepts to model hierarchical system architectures, covering both software components and hardware resources, because most NFPs base on the software's interaction with hardware resources (e.g., execution time and energy consumption). QCL provides concepts to express dependencies between CCM components based on NFPs. This implies dependencies between software components as well as software and hardware components. In the following, we introduce CCM and QCL, by means of a video application scenario.

A. Capturing Software and Hardware Components

The CCM distinguishes between modeling the system structure of hardware resource types, software components and variants of both. In the scope of this paper, variants of resource types are concrete hardware resources; variants of software components are concrete implementations. The system structure defines how a system looks like and, thus, represents type declarations for specific variants. For instance, consider the upper part of Figure 1 showing the types for a video application. It consists of three software components, namely a `Player`, a `Decoder` and a `DataProvider`. Each component may have one or more port types representing interfaces of the component. Port types can be used to connect different components. A set of connected components describes a software system.

Concrete implementations of software components (cf. Figure 1) have to correspond to their type. In the given example two variants of `Player`s, the `VLC` (Video LAN Client) and Quicktime (`QT`) implementation exist.

 ISBN: 978-1-61208-274-5

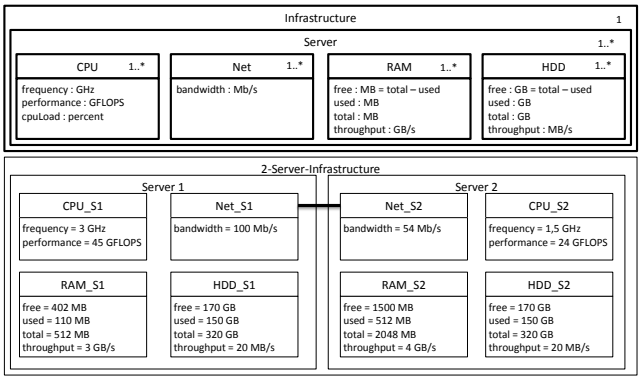


Fig. 2. Top: CCM Structure Model for hardware landscapes. Bottom: CCM Variant Model for a hardware landscape of 2 servers.

```
1  contract VLC implements VideoPlayer {
2    mode highQuality {
3      //required resources
4      requires resource CPU {
5        max cpuLoad = 50 percent
6        min frequency = 2 GHz
7      }
8      requires resource Net {
9        min bandwidth = 10 MBit/s
10     }
11     //dependencies on other SW components
12     requires component Decoder {
13       min dataRate = 50 KB/s
14     }
15     //what is provided in turn
16     provides min frameRate 25 Frame/s
17     provides min resolution 1080 p
18   }
19   mode lowQuality { ... }
20 }
```

Fig. 3. Example Contract for VLC Video Player.

For `Decoder`s, a free (`Free`) and a commercial (`Com.`) implementation are available. Finally, the `DataProvider` is implemented as a local file reader (`File`) and a remote URL reader (`URL`).

To capture types available in the hardware landscape, resource types are specified. The upper part of Figure 2 defines resource types, on which our video application shall be executed. The `Infrastructure` consist of one or more `Server`s, whereas each server contains one or more `CPU`s, network interfaces (`Net`), `RAM` chips and hard disks (`HDD`). For reasons of simplicity, we omit port types of resource types in the given example.

For each component type (software and hardware), NFPs can be defined. For instance, the `Player` type defines a property `framerate` in fps (frames per second) whereas the `HDD` type defines a property `used` (disk space) in GByte.

The lower part of Figure 2 shows a hardware instantiation of the resource type system mentioned above. It consists of two servers with specific resources according to the definitions at the type level. NFPs defined at type level, are available at variant level with concrete values. Each resource variant can provide behavior models to further specify its NFPs (e.g., the correlation of energy consumption and CPU utilization).

B. Specification of QCL contracts

QCL is used to define dependencies between CCM components as contracts, specified for each variant. Therefore, a QCL contract represents a specific view on a variant regarding its dependencies to other types. A contract defines one or more `mode`s, whereas each `mode` indepently (from other modes) defines dependencies to other components. Software components can depend on other software components as well as hardware resources, whereas resources can depend on other hardware resources only. Each dependency relates to a component type and defines bounds for required values of NFPs at runtime. In addition to property requirement constraints, provided NFPs are specified as well.

Figure 3 shows a contract for the *VLC* video player as a concrete implementation of the `VideoPlayer` component. As defined, the player can be used in two modes: `high-` and `lowQuality`. For `highQuality` the contract specifies requirements for a `CPU` and a `Net` device. The `CPU` needs to be utilized at most to 50% and to have a frequency of at least 2GHz. The `Net` device has to offer at least a 10 MBit/s bandwidth. Furthermore, another component is required—a `Decoder`. Any implementation of this type, which is able to provide a data rate of at least 50 KB/s can be used. Finally, the contract defines that in the `highQuality` mode a minimum framerate of 25 fps and a resolution of 1080p is provided.

To determine the hardware requirements, micro-benchmarks written by the component developer, evaluating the non-functional properties of interest, are used. A more detailed discussion on how to create these contracts has been published in [8].

In summary, a system modeled with CCM and QCL is highly variable in terms of multiple implementations of component types, multiple quality modes per implementation and, according to resource requirements of each quality mode, multiple possible mappings of implementations to hardware resources.

III. Self-Optimization with ILP

The central task of self-optimizing systems is to determine optimal system configurations. In this section, a model-based approach using an exact optimization technique called ILP is shown. In contrast to many existing approaches to self-optimization, the presented approach does not require the developer to apply the optimization technique itself, but generates the formulation of the optimization problem using the existing development artifacts. In this work, a system configuration denotes a set of software component implementations deployed on component-containers, which run on servers (or, more general, computing entities). Thus, the optimization problem is, which implementations of which component types need to be mapped onto which containers in order to reach the optimal trade off between user satisfaction and execution costs.

To solve this problem, a variety of information is re-

 ISBN: 978-1-61208-274-5

quired. Namely, variant models of hard- and software representing the currently running system, structure models of hard- and software representing the architecture of the system, QCL contracts characterizing the non-functional behavior of the software and a user request with the user's Quality of Service (QoS) demands.

As ILP is a mathematical formalism with its own language, a transformation from the structure and variant models as well as the contracts and the request to ILP is required. Figure 4 depicts the general approach of this ILP generation, which can be characterized as a model-to-text transformation in terms of Model-Driven Architecture (MDA) [9].

On the left upper side of Figure 4, the structure of the optimization problem formulated as an ILP is shown. An ILP comprises a set of objective functions, a set of decision variables and a set of constraints (as highlighted by the dashed lines). The objective functions depend on the user request (declaring objectives important for the user, e.g., response time) and on the variant (i.e., runtime) model. Decision variables depend on QoS contracts and variant models, too. Finally, the constraints of the ILP depend on all available input information. In the following, the generation of decision variables, objective functions and constraints is discussed in more detail.

A. The Rational of Decision Variables

In this work, the decision variables directly follow the characterization of system configurations: they denote which implementation is to be run in which quality mode on which container. They encompass a selection problem (i.e., which implementations to select) and a mapping problem (i.e., to which container the selected implementations shall be mapped). This information is comprised by the name of the variable seperated by hashtags, whereas the type of the decision variables is of boolean nature (meaning each of these variable being solved having the value `1` denotes the deployment of a specific component implementation in a specific mode on a specific resource). Equation 1 shows the general form of decision variables as used in the presented approach. The prefix `b#` is meant to highlight the boolean type of this variable.

$$b\#implementation\#mode\#container \in \mathbb{B} \quad (1)$$

Additional variables express the resource utilization and resulting NFP values implied by a certain implementation (as specified in QCL contracts). These variables have a real value (i.e., are in $\mathbb{R}$) and have the prefix `u#` for utilization. The naming of these variables fully qualifies an NFP of a certain resource of a component container (i.e., server). As resources are hierarchically structured, subresources, subsubresources, etc. can be specified. For example, a resource `CPU1` can comprises the subresources `Core1`, `Core2` and so on. Equations 2 and 3 denote the general forms of these variables.

$$u\#container\#resource\#subres.\#...\#NFP \in \mathbb{R} \quad (2)$$

$$implementation\#NFP \in \mathbb{R} \quad (3)$$

Solving an ILP working on these variables, leads to an assignment of values, representing the optimal system configuration. Thus, the solution provides information about the optimal selection of implementations, modes, their mapping to containers and additional information on the resulting NFPs of the participating components.

B. Generation of Objective Functions

All objective functions in the context of this work, base on assessment functions of system configurations. That is, an objective function assesses system configurations in terms of the respective objective (e.g., energy, performance or reliability) and aims to determine that configuration, which is assessed to be minimal or maximal w.r.t. the current objective.

A straightforward objective function based on the variables explained in the previous subsection is resource minimization as shown in Equation 4. However, this objective function does not consider the interplay between selected components instances there, required and provided NFPs and the available resources. Thus, constraints are used to restrict the ILP to correct solutions. Anyhow, the objective function of Equation 4 does not lead to the intended result (i.e., minimum resource consumption), because the units and the semantics of each resource usage variable are not considered. For example, the formula does not differentiate between utilizing 10 MB of main memory in contrast to utilizing 10 MB of hard disk drive (HDD) space.

$$\min : \sum u\#container\#res.\#subres.\#...\#NFP \quad (4)$$

A practical solution towards more sophisticated objective functions is the application of utility theory to map each variable to a utility expressed as a real value between zero and one. In the case of resource usage the utility functions can reflect the difference between using space of main memory and an HDD by putting the requested amount of space in relation to the totally available space. The general form of objective functions according to utility theory is shown in Equation 5. The objective is to maximize the overall utility.

$$\max : \sum utility(var_{decision}) \quad (5)$$

An even more sophisticated objective is the combination of the previous two types of objectives (i.e., cost minimization and utility maximization): *efficiency* maximization. The general form of this type of objective is depicted in Equation 6.

$$\max : \eta(var_{decision}) = \frac{utility(var_{decision})}{cost(var_{decision})} \quad (6)$$

Finally, an important aspect of objective functions in the context of reconfigurable systems is the need to consider the **reconfiguration** and **decision making** itself.

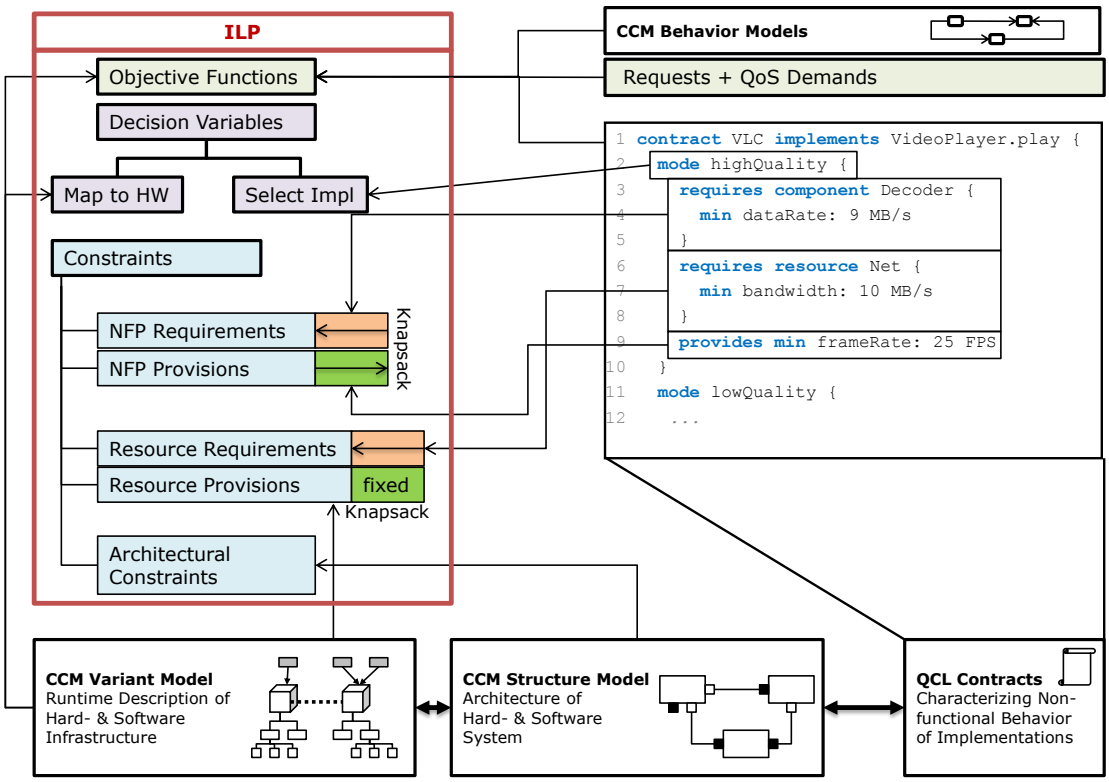


Fig. 4. Overview of ILP Generation.

This is, because both processes require time and resources and, thus, effect the objective functions. Hence, a general objective function should look as depicted in Equation 7, which aims for maximum efficiency. Notably, the costs implied by reconfiguration and decision making need to be assessed w.r.t. the quality of interest for the respective objective function.

$$\max : \sum_{i=1}^{n} \frac{util_i}{cost_i} * \frac{util_{reconf} + util_{decide}}{cost_{reconf} + cost_{decide}} \quad (7)$$

The assessment of $cost_{reconf}$ as well as $cost_{decide}$ (and the negative utilities) is a non-trivial task, which has to be investigated for each NFP individually. A closer discussion exceeds the scope of this paper.

C. Constraint Generation

As mentioned above, the ILP must be constrained by a set of constraints to allow the computation of sensible solutions only. In general, three classes of constraints are generated in the presented ILP approach: constraints negotiating software NFPs, constraints negotiating resource requirements, and architectural constraints. In the following each class is explained in more detail.

Software NFP Negotiation: The negotiation of software NFPs covers the interdependencies between NFPs of different software components expressed by their provisions and requirements in QCL contracts. The selection of an implementation of a software component type in a certain quality mode induces a set of NFP provisions (by the implementation) and requirements (to other components). To determine an optimal selection includes to find a balance between provided NFPs and required NFPs across all required components to fulfill the user's request. This problem is reflected in the ILP by two types of constraint clauses, which are generated for each NFP. First, NFP provisions are expressed as equality constraints as depicted in Equation 8. Depending on the assignment of the decision variables, the available amount of the respective NFP results.

$$NFP_i = \sum^{c} prov_a^b * b\#impl_a\#mode_b\#container \quad (8)$$

Second, the NFP requirements implied by selecting a certain implementation in a quality mode are expressed as inequality constraint as depicted in Equation 9. Notably, the relation between NFP_i and the aggregated NFP requirements is only "less or equal", if the respective NFP is of ascending order. For example, the NFP `memory` is of ascending order, whereas the NFP `response_time` is of descending order. Thus, for `response_time`, the inequality is of "greater or equal than" type.

$$NFP_i \leq \sum^{c} req_a^b * b\#impl_a\#mode_b\#container \quad (9)$$

 ISBN: 978-1-61208-274-5

Resource Negotiation: Besides NFPs provided and required by implementation, the selection of implementations implies resource requirements, too. In contrast to software NFPs, the provision of resources is fixed by the available hardware. Similar to software NFP negotiation, two types of constraints are generated for resource negotiation. First, the provision of resources as depicted in Equation 10. The provided property naturally needs to be greater or equal than zero and less or equal than the maximum offered by the resource. In addition, the granularity of the resource can be restricted. For example, the amount of disk space can only be utilized in blocks of a certain size (e.g., 4KB). In the constraints of Equation 10, the terms *maximum* and *granularity* are replaced by the respective concrete values. The term x remains a free variable, which is to be found by the solver of the optimization problem. The restriction of x to be binary (i.e., $\in \mathbb{B}$) enforces the restriction of the resource property to be a multiple of *granularity*.

$$\begin{aligned} ResourceProperty_i &\geq 0 \\ ResourceProperty_i &\leq max \\ ResourceProperty_i &= granularity * x \\ x &\in \mathbb{B} \end{aligned} \tag{10}$$

The second type of constraint covers the resource requirements by selecting an implementation. The resulting constraint is depicted in Equation 11.

$$ResourceProperty_i \leq \sum^{c} req_a^b * b\#impl_a\#mode_b\#container \tag{11}$$

For each resource property, constraints of these types are generated, whereby the search for valid assignments to the decision variables is further restricted.

Architectural Constraints: Finally, constraints based on the knowledge about the software's architecture and the requested software component are translated into constraints of the ILP. The simplest possible constraint of this type is the necessity to select exactly one implementation of a component type whose port is requested by the user. The corresponding constraint is depicted in Equation 12.

$$\sum b\#impl_a\#mode_b\#container = 1 \quad \forall a \in T \wedge b \in modes_of(a) \tag{12}$$

For all modes b of all implementations available for the component type T, the sum of the corresponding decision variables needs to be exactly one. This constraint suffices, if no other component types exist or the requested component type does not use any other component type. If another component type is used, the need to select an implementation of this type needs to be expressed, too. In the general case, constraints have to be generated, which express that the selection of an implementation of component type T_1 implies the need to select an implementation of type T_2. Equation 13 depicts this kind of constraint.

$$\begin{aligned} \sum b\#impl_a\#mode_b\#container &= \\ \sum b\#impl_c\#mode_d\#container & \\ \forall a \in T_1 \wedge b \in modes\_of(a) \wedge c \in T_2 &\ \wedge \\ d \in modes\_of(c) \wedge depends(T_1, T_2) & \end{aligned} \tag{13}$$

The above described three types of constraints, restricting the possible assignments to the decision variables, so only valid configurations are investigated for their optimality. In addition, corresponding to the decisions, values are assigned to the resource usage and NFP variables. In the next section we describe a lean variant of this approach, which avoids the use of resource and NFP variables.

IV. Self-Optimization with PBO

The key aspect of the configuration problem expressed for the ILP-based solution presented above, is denoted by boolean decision variables, encompassing the decision to select certain implementations and their mapping to certain resources. The remaining variables used in the ILP-based solution comprise resource usage and resulting NFP values. In this section, the ability to omit non-boolean variables is shown. The intended goal is to apply more efficient solving techniques to the generated optimization problems, which leverage on the restriction to use boolean variables only. Due to the exclusive use of boolean variables in an ILP a special type of ILP results: a 0-1 ILP. This type of ILP can be handled by PBO, which applies techniques used to solve satisfiability problems in propositional logics (e.g., DPLL [10]). These techniques have polynomial complexity, whereas algorithms used to solve ILPs (e.g., simplex) have exponential complexity. Hence, we investigate PBO for self-optimization.

A. Reformulation of the Configuration Problem in PBO

To apply techniques of PBO to the configuration problem to be generated, all non-boolean variables from the ILP solution need to be expressed in a different way. Namely, a new way to express resource negotiation, NFP negotiation including user requirements and the objective function is required. Notably, the architectural constraints defined for the ILP solution can remain unchanged, because they only refer to decision variables. In the following, a solution for each of the constraints subject to adjustment is given.

Resource Negotiation without Usage Variables: The expression of resource negotiation in ILP with usage variables has been shown in the previous section. All these constraints, except for the granularity restriction, can be expressed by a single PBO constraint, which implicitly represents the respective resource property (i.e., $ResourceProperty_i$) as shown in Equation 14. The term ${}^c req_a^b$ denotes the implied resource requirements by the respective implementation a in the specified mode b on a given container c.

$$\sum {}^{c}req_a^b * b\#impl_a\#mode_b\#container_c \leq max \quad (14)$$

NFP negotiation: Alongside with the restriction of resource usage, the dependencies between offered and required NFPs has to be expressed by constraints. In the ILP solution explicit variables for each NFP have been used to connect separate constraints for their provisions and requirements. The same principle, as for resource usage negotiation can be applied for NFP negotiation, too. Namely, the implicit expression of each NFP variable. Thus, for PBO, the provision and requirement constraints are merged into a single constraint which implicitly represents the possible expressions of the NFP. For the generation of these constraints the user request needs to be considered, too. This can be accomplished by incorporating the respective NFP requirements expressed by the user in his request as minimum value of the respective NFP. The resulting constraint is shown in Equation 15.

$$\begin{aligned}\sum (req_{user} +^{c} req_a^b * b\#impl_a\#mode_b\#container_c) \quad (15)\\ \leq \sum {}^{c}prov_a^b * b\#impl_a\#mode_b\#container_c\end{aligned}$$

Reformulation of Objective Function: The objective function in PBO can only rely on decision variables, because only these variables exist. In contrast, the ILP solution allowed for the application of resource usage and NFP variables. Thus, the minimization or maximization of resource usage and NFPs cannot be directly expressed in PBO. The general form of the objective function in PBO is shown in Equation 16.

$$min : \sum weight * b\#impl_a\#mode_b\#container_c \quad (16)$$

Thus, the major issue to generate meaningful objective functions in PBO is the computation of the *weight* constants for each decision variable. This *weight* has to express the impact the decision represented by the decision variable on the overall objective.

V. Evaluation

The above described approach, applying generated ILP- or PBO-based optimization for self-adaptive systems, is *not likely to scale*. This is, because solving an ILP or PBO is known to be an NP-hard problem, where the processing time required by the solver grows exponentially with number of decision variables of the ILP/PBO. In the following, we show how ILP/PBO generation and solving perform and that both approaches are feasible for typical pipes-and-filter applications. We compare the performance of both approaches, showing that the ILP solution outperforms the PBO solution.

A. Generation of Test Systems for Empirical Evaluation

To empirically evaluate the performance of the approaches, a set of test systems has been generated. As the ILP/PBO generation relies on the models of the system only (and not the system itself), it is possible to evaluate the approaches against a variety of system types without the need for their physical presence. Thus, we developed a parameterizable system generator, which is capable of generating models as usually derived by the runtime environment. This includes hard- and software structure models, hardware and software variant models and QoS contracts.

The generator is configured with several parameters:

- Number of servers S,
- Number of resources per server N_{res},
- Number of properties per resource $N_{resProp}$.
- Number of component types, C
- Maximum depth of dependency chains δ,
- Number of NFPs defined per component type N_{nfp},
- Number of implementations per component type N_{impl},
- Number of modes per implementation N_{mode},
- Number of hardware requirements per mode,
- Number of NFPs required per software dependency per mode and
- Number of provided NFPs per mode.

Note that it is impossible to generate systems leading to worst case execution times of the solver as the solvers internally use heuristics. But, for proper evaluation results using generated systems, each generated system should allow at least one valid configuration. Randomly generating NFP provisions and requirements obviously leads to infeasible systems in most cases. To ensure the existence of at least on feasible system configuration, a random request is generated, which serves as reference request for which a feasible system configuration is to be ensured. The process of system generation keeps track of how the random request transforms between dependent software component types. For the directly requested component type at least one (randomly chosen) quality mode Q_1 is selected to fulfill the request. Then, for all dependent component types at least one quality mode is chosen to fulfill the requirements of Q_1. The same process is performed for all dependent types of the dependent types and so on. The generated user request to ensure feasibility is reusedlater as test request for evaluation.

B. Measurements for Pipe-and-Filter Style Systems

The pipes-and-filter architectural style is common across many data processing applications. For example, in early detection of Alzheimer's disease, magnetic resonance tomography (MRT) pictures are processed by a

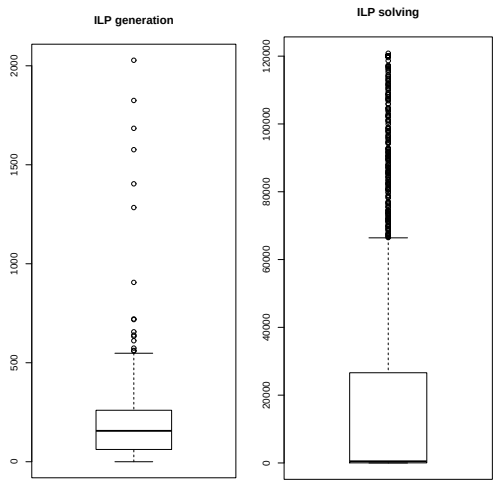
pipes-and-filter architecture comprising data preparation, Alzheimer's indicator search and data postprocessing. Another example is audio processing as performed, e.g., by auphonic (http://auphonic.com/), where audio files are processed by a chain of processing steps. In general, the pipes-and-filter style is characterized by a chain of component types. There are n component types, where the first component type requires the second, which in turn requires the third component type and so on. For this architectural style the parameter n, denoting the number of component types or the depth of the chain is of interest. In addition, the number of containers c is to be considered, because the number of possible configurations grows exponentially with the number of available containers. Thus, a test system in pipes-and-filter style is characterized as an $n \times c$ system having n component types and c containers.

For the server landscape, we assume each server (i.e., container) to have one central processing unit (CPU), one random access memory (RAM) module, on HDD and a network device. Each software component type has one provided and one required port type, except the last component type in the chain, which only has a provided port type. Moreover, each software component type has two NFPs and two implementations, having two modes, which each have four resource requirements, two software requirements and two provisions.

To assess pipes-and-filter type applications, all variants of C *x* S systems for $C = [2..100]$ and $S = [2..100]$ have been generated and the generation and solving time of the respective ILPs and PBO formulations has been measured. For each generated system, two measurements were taken. First, the time required to generate the respective ILP or PBO problem. Second, the time required to solve the problem using a standard solver. For the ILP solution the solver *LP Solve* [11] (version 5.5.20) has been used. For PBO the *OBPDB* [12] solver (v. 1.1.3) has been used. All measurements were taken on a DELL Alienware X51 desktop PC running Windows 7 64bit and containing a solid state disk, 8 GB DDR1600 RAM and an Intel Core i7-2600 CPU running at 3.4 GHz having 4 physical cores and 8 virtual cores by hyperthreading.

Analysis of the ILP Solution: Figure 5(a) shows a boxplot of the ILP generation time and Figure 6 depicts this generation time in relation to the number of software components. The median generation time is 156 ms and 75% of all ILPs were generated in at most 260 ms. The longest generation took 2028 ms. Notably, the 99%-quantile is 437 ms, meaning that in 99% of all cases, the maximum generation time is less or equal to 468 ms. A natural hypothesis is that the number of components and servers correlates to the generation time; which indeed exists: $T_{gen}(C) = 0.0291C^2 + 1.4429C + 5.3851$ with an R^2 of 0.8956.

Figure 5(b) depicts a boxplot of the ILP solving time. It reveals the random nature of worst and best case runtime of ILP solving, depicted by the vast amount of outliers. For only 121 out of 9801 generated ILPs the solver was not able to return any solution within two minutes for all measurements taken (i.e., in 1.2% of all cases). Please note, that the solving time had an upper limit at 2 minutes, as



(a) Generation time (b) Solving time

Fig. 5. ILP generation and solving time in milliseconds.

for a some ILPs the solving time can increase to multiple hours or even days, which is the worst case, where the complete problem space has to be explored. Surprisingly, the median solving time is only 478.5 ms. Thus, half of the ILPs could be solved in less than a second. The third quantile (i.e., 75%-quantile) is at 26.58 s. 79% of all ILPs were solved in less than a minute. Notably, if the (manually configured) timeout of two minutes was reached, the ILP solver returned the best solution found so far.

In the following, a closer investigation of the solving time is presented. The aim is to investigate if and how

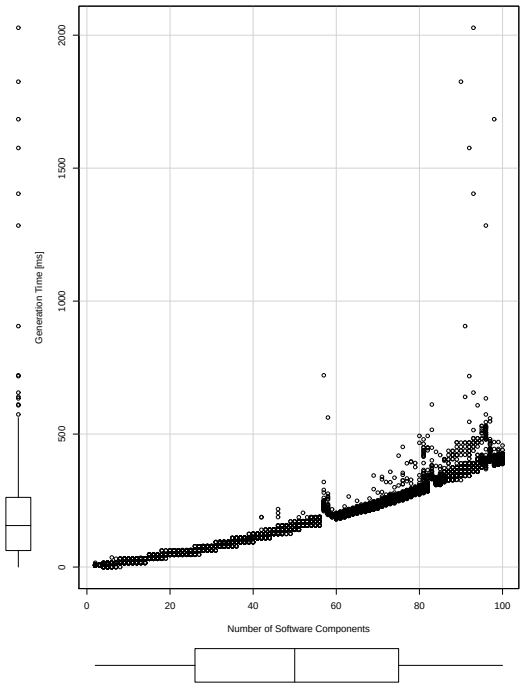


Fig. 6. ILP generation time in relation to number of components.

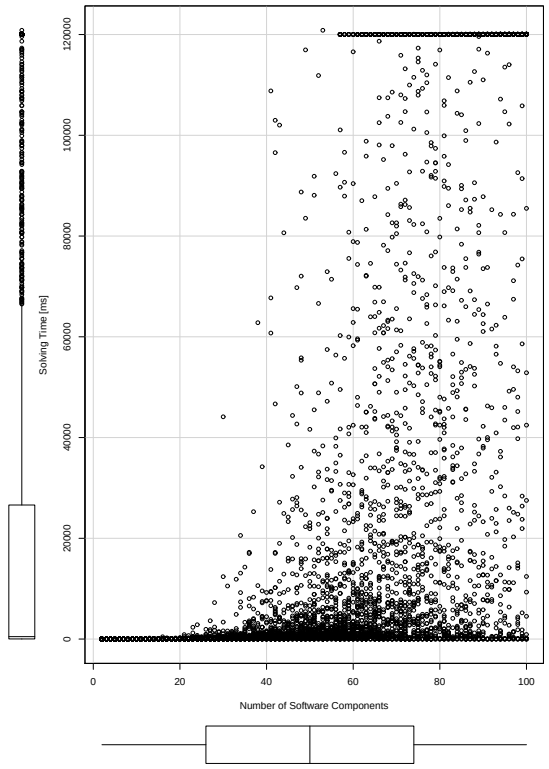


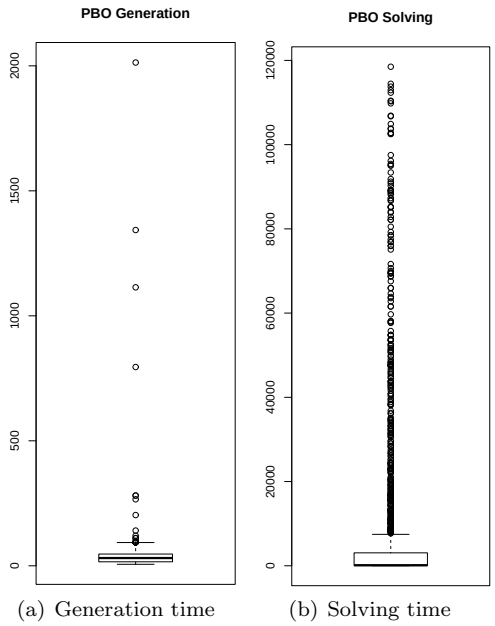
Fig. 7. ILP solving time in relation to number of components.

the system parameters correlate with the solving time of the generated ILP. The hypothesis is, that there is a correlation between the number of component types and the solving time. Figure 7 depicts this correlation. On each axis a boxplot of the corresponding variable is shown to highlight the high density of solutions in low solving times. An interesting conclusion from this figure is, that the predictability of solving time decreases at approximately 25 component types. Most ILPs are solved in a few seconds, though the more component types, the more likely are longer solving times.

The correlation between the number of component types and the solving time for scenarios is statistically poor. The linear regression has an adjusted R^2 of only 0.4286. Exponential regression (i.e., $T_{solve} = f(Components) = a \cdot e^x$) reveals a residual standard error of $1.911 \cdot 10^{13}$. Thus, there is no statistically significant correlation between the number of components and the solving time. The reason is the random nature of ILP solving, i.e., solving random ILPs can randomly lead to worst and best case situations.

Analysis of the PBO in Comparison to the ILP Solution: The same measurements have been done for the PBO solution. Figure 8 depicts the PBO generation and solving times. In addition, Figure 9 depicts the generation time in relation to the number of components. Please note that for the PBO solution only systems with up to 30 component types have been measured, because the approach does not scale beyond this number of component types.

In comparison to the ILP solution, the generation of



(a) Generation time (b) Solving time

Fig. 8. PBO generation and solving time in milliseconds.

PBOs looks comparably performant at a first glance. ILPs for systems of up to 100 component types are generated in up to 2 seconds. PBOs for systems of only up to 30 component types require up to 2 s, too. But, whilst the median for ILP generation was at 156 ms, for PBO generation it is at 31 ms. Also for the 99%-quantile, the ILP solution looks worse than the PBO solution, as for ILP the 99%-quantile is at 468 ms, whereas for PBO it is at 94 ms.

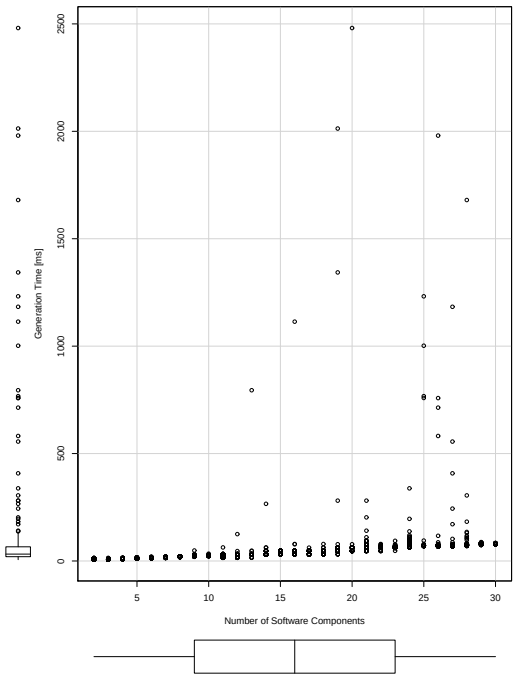


Fig. 9. PBO generation time in relation to number of components.

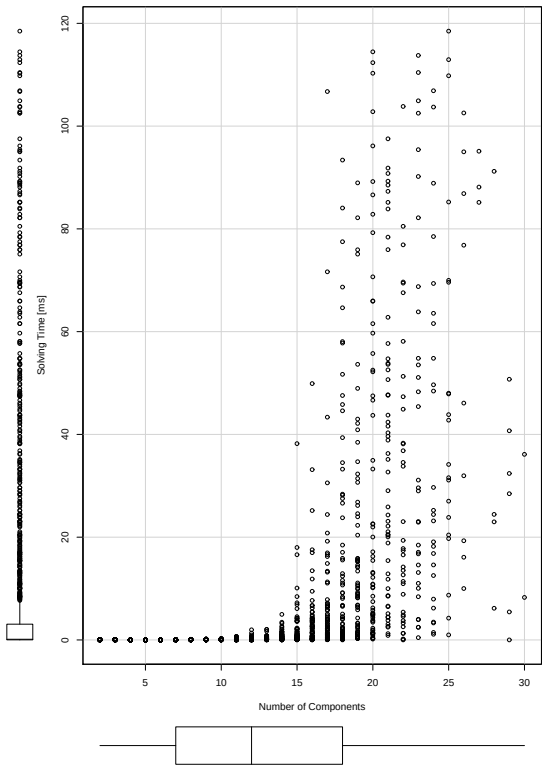
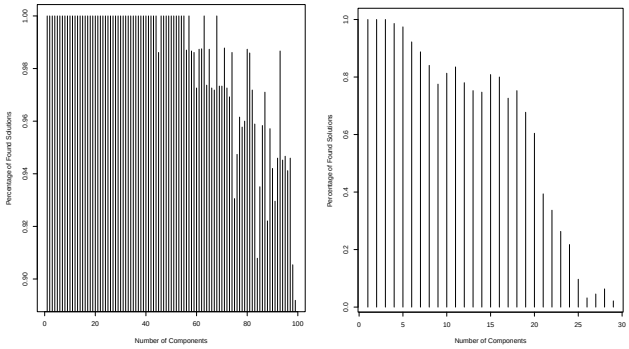


Fig. 10. PBO solving time in relation to number of components.

But, because for the PBO solution only systems of up to 30 component types have been used, the ILP generation times for up to 100 components cannot be used as a reference. An investigation of ILP generation for systems of up to 30 component types reveals a median of 32 ms and a 99%-quantile of 78 ms. Thus, PBO generation is even slower than ILP generation, although less constraints and less variables have to be generated. The reason for the better performance of ILP generation is the required program format for the applied solvers. The ILP solution can use long variables names to encode information (i.e., the decision variables encode their meaning in their name), whereas the PBO solution has to use enumerated variables (i.e., x_n). In consequence, the PBO generation has to handle the mapping of decision variables to their short versions, which consumes time and, hence, leads to the measured performance loss.

An investigation of the solving time of the PBO solution reveals surprising results. The rational for using PBO instead of ILP was the hypothesis that PBO performs better in many cases as it has polynomial complexity only. But, for the optimization problem discussed in this paper it apparently does not as Figure 10 depicts. The solving time has been limited to two minutes, too. But, in contrast to the ILP solution, the PBO solver does not deliver a suboptimal solution if the timeout is reached. Starting at 15 component types (compared to 25 in the ILP solution) the predictability of the solving time drastically decreases. Most interesting is the lack of fast solutions starting at 25 component types. In comparison, the ILP solution was able to deliver solutions in a few milliseconds even for



(a) ILP approach (b) PBO approach

Fig. 11. Percentage of found solutions for ILP and PBO within 2 minutes of solving time.

systems with more than 80 component types. Moreover, the number of determined solutions in a timeframe of less than 2 minutes significantly reduces starting already at 20 component types. For example, the measurements taken for 28 component types and 2 to 100 servers (i.e., 99 measurements) only lead to 4 solutions. For the remaining 95 systems the PBO solver could not find a configuration. For this reason, only measurements for up to 30 component types have been collected.

The measurements of the solving time indeed benchmark the used solver. Unfortunately, for PBO only one solver could be investigated, because all other solvers do not support the specification of equalities with variables on both sides of the equation. Therefore, OPBDP [12] was the only publicly available, working solver which could be investigated.

Thus, in conclusion, the PBO solution performs much worse than the ILP solution for the optimization problem at hand. This is because, *in the average case* the algorithms used to solve ILPs perform better than those used for PBO. Whereas the ILP solution is feasible for systems of up to 100 component types, the PBO solution can handle at most 25 component types. The generation of ILPs is below 437 ms for up to 100 component types and PBO generation takes less than 100 ms in 99% of all cases for systems of up to 30 component types. Solving of ILPs is below 500 ms in 50% of all cases and below 27 s in 75% of all cases. PBO solving is below 2.6 s in 50% of all cases, but reaches the timeout of 2 minutes already in 70% of all cases. Most notably, the PBO solution is not able to find configurations starting already at 25 component types in most of the cases, whereas the ILP solution is able to determine configurations even for 100 component types. Figure 11 depicts the percentage of solutions found by the ILP and PBO approach in correlation with the number of component types. The execution time of the ILP solution is predictable up to 25 component types, whereas the execution time of the PBO solution only up to 15 component types.

The reason for the decreasing predictability in the ILP solution is the growing span between best and worst case execution time of the solver. The best case execution time grows linearly with the number of variables, whereas the

 ISBN: 978-1-61208-274-5

worst case execution time grows exponentially. Both curves are very near to each other in the beginning, but depart more and more (exponentially) from each other the bigger the systems are. Notably, the numbers presented above are specific to the machine used for measurements. Thus, also the number of component types where predictability starts to worsen has to be determined for each machine.

VI. Related Work

The application of model-driven software development to self-adaptive systems has been studied by various groups throughout the last years.

In [13], Zeller et al. address the problem of determining a valid mapping of software components to electronic control units (ECU) by SAT solving and simulated annealing. The solving techniques do not guarantee optimality, but ensure the determination of a valid system configuration. The SAT problem is generated from the system's models. Zeller et al. evaluated their approach and showed that the runtime of their SAT solving approach is below 4s for 40 ECUs, 60 Sensors, 60 Actuators and 2000 Functions. For less than 1600 functions SAT solving takes less than 2 seconds. The biggest setup evaluated by Zeller et al. comprised 100 ECUs, 120 Sensors, 120 Actuators, 2500 Functions and took 18 seconds [13].

Another approach using model-driven software development for self-adaptive systems has been developed in the DiVA research project and presented in [14], where Fleurey and Solberg introduce a quality grading framework for software variants. The developer rates available implementations in different quality dimensions (e.g., energy or performance) using a model-based framework. To identify valid system configurations these models are transformed to Alloy [15]. Unfortunately, no empirical study of the approach's scalability has been published.

In contrast to the approaches presented in this paper, the previously discussed approaches do not search for optimal configurations, the constraints of the optimization problem are not generated, the notion of contracts is not utilized and only the mapping [13] or the selection problem [14] is considered, respectively.

VII. Conclusion and Future Work

In this paper, we propose the application of model-driven software engineering and generation of optimization problems to automatically compute optimal system configurations for self-adaptive software systems. We presented two approaches, an ILP-based and a PBO-based solution. The rational for investigating the applicability of PBO was the hypothesis that PBO could perform better than ILP for the investigated optimization problem. However, based on a large set of systems to be optimized, our scalability analysis rejected this hypothesis and revealed the feasibility of the ILP solution for systems of up to 100 component types to be distributed on 100 servers. As typical data processing applications like audio processing usually comprise a few tens of processing steps, the ILP-based approach has been shown to be applicable. For future work, we plan to investigate the feasibility and scalability of further optimization techniques.

Acknowledgment

This research has been funded by the ESF and Federal State of Saxony within the project ZESSY #080951806, and within the Collaborative Research Center 912 (HAEC), funded by the DFG.

References

[1] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in Future of Software Systems, 2007, pp. 259–268.

[2] B. H. Cheng, R. Lemos, and H. Giese et al., "Software engineering for self-adaptive systems: A research roadmap," in Software Engineering for Self-Adaptive Systems, ser. LNCS. Springer, 2009, vol. 5525, pp. 1–26.

[3] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@Run.time to Support Dynamic Adaptation," Computer, vol. 42, no. 10, 2009, pp. 44–51.

[4] E. Boros and P. L. Hammer, "Pseudo-Boolean Optimization," Discrete Applied Mathematics, vol. 123, no. 1–3, November 2002, pp. 155–225.

[5] G. L. Nemhauser and L. A. Wolsey, Integer and Combinatorial Optimization. Wiley Interscience, 1999.

[6] S. Götz, C. Wilke, M. Schmidt, S. Cech, and U. Aßmann, "Towards Energy Auto Tuning," in GREEN IT 2010. GSTF, 2010, pp. 122–129.

[7] S. Götz, C. Wilke, S. Cech, and U. Aßmann, Sustainable ICTs and Management Systems for Green Computing. IGI Global, 2012, ch. Architecture and Mechanisms for Energy Auto Tuning, pp. 45–73.

[8] S. Götz, C. Wilke, S. Richly, and U. Aßmann, "Approximating quality contracts for energy auto-tuning software," in GREENS 2012. IEEE, 2012, pp. 8–14.

[9] J. Miller and J. Mukerji, "MDA Guide Version 1.0.1," OMG Document, 2003.

[10] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," CACM, vol. 5, no. 7, Jul. 1962, pp. 394–397.

[11] K. Eikland and P. Notebaert, "LP Solve 5.5 Reference Guide," http://lpsolve.sourceforge.net/5.5/ (access in Nov. 2012).

[12] P. Barth, "A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization," Max-Planck-Institut für Informatik, Research Report MPI-I-95-2-003, 1995.

[13] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr, "Towards Self-Adaptation in Real-time, Networked Systems: Efficient Solving of System Constraints for Automotive Embedded Systems," in SASO 2011, 2011, pp. 79–88.

[14] F. Fleurey and A. Solberg, "A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems," in MODELS 2009, ser. LNCS, vol. 5795. Springer, 2009, pp. 606–621.

[15] D. Jackson, Software Abstractions – Logic, Language, and Analysis. MIT Press, 2012.

 ISBN: 978-1-61208-274-5

Towards Systematic Model-based Testing of Self-adaptive Software

Georg Püschel, Sebastian Götz, Claas Wilke, Uwe Aßmann
Software Technology Group, Technische Universität Dresden
Email: {georg.pueschel, sebastian.goetz, claas.wilke, uwe.assmann}@tu-dresden.de

***Abstract*—Self-adaptive software reconfigures automatically at run-time to environment changes in order to fulfill its specified goals. Thereby, the system runs in a control loop which includes monitoring, analysis, adaptation planning, and execution. To assure functional correctness and non-functional adequacy, testing is required. When defining test cases, the control loop's tasks have to be validated as well as the adapted system behavior that spans a much more complex decision space than in static software. To reduce the complexity for testers, models can be employed and later be used to generate test cases automatically—an approach called Model-based Testing. Thereby, a test modeler has to specify test models expressing the system's externally perceivable behavior. In this paper, we perform a Failure Mode and Effects Analysis on a generic perspective on self-adaptive software to figure out the additional requirements to be coped with in test modeling.**

Keywords*—*self-adaptive software; problem statement; model-based testing; failure mode and effects analysis

I. INTRODUCTION

Self-adaptive software (SAS) reconfigures automatically at run-time according to sensed environment changes. Thus, it is able to effectively and efficiently fulfill its specified goals in the changed conditions. SAS runs in a control loop that frequently operates four tasks: monitoring, analysis, planning, and execution (MAPE, [1]).

Quality assurance for SAS includes validation or verification of the control loop's tasks. While, for instance, run-time self-testing of SAS was researched (e.g., in [2]), there still is a lack of black box testing approaches. However, such concepts are or will be required by the software industry, e.g., due to the need for certification. Systems are delivered and left to the customer and have to be reliable and evaluated in advance as far as possible under the respective project conditions. Hence, not only fault-tolerance but also fault-prevention is desirable.

The crucial challenge of SAS black box testing [3, p. 17] is to handle its complexity. The behavioral decision space is extensively expanded due to the impact of adaptation on adapted system structures and interacting running processes. A further problem is *error propagation*: errors produced in one component can be transformed into errors which manifest in other components [4] and—due to the looped control flow—as a permanent inner system state. Thus, manifold information have to be considered by testers, injected into the tested system, monitored, and evaluated for determination of behavioral correctness. In consequence, *manual* test case definition for an SAS is mostly hard to manage for test engineers.

In constructive phases of SAS engineering, this complexity is dealt with by using models. Modeling takes advantage of abstraction and convention (i.e., implicit information) to hide details from designers. For testing, an equivalent concept was developed: in *model-based testing* (MBT, [5]) the system under test's (SUT) interfaces are structually and behaviorally modeled and later test cases are automatically generated. For limitation and measurement of actually tested parts of the behavioral space, a test coverage criterion can be specified. Due to the focus on the SUT's interface, MBT is a *black box* approach.

As we previously constructed test models for run-time variable mobile systems [6], we are now focusing on generalization of our experience and provide reasonable test models for SAS. A prerequisite for the application of MBT methods is to gather the following information:

1) Scenarios, how failures can occur, have to be found such that the progress of system validation be can estimated, effort predicted, and testers are aware of the potential complexity of their task.
2) Properties that failures can comprise have to be identified to provide meaningful verdicts.
3) Potential error propagation has to be investigated to identify causal chains.

While the above points fit on arbitrary system types, it becomes necessary to find more specific test modeling requirements for SAS. To bypass the intuitive formulation of these requirements we decided to perform a systematic analysis of critical failure properties and scenarios based on *Failure Mode and Effects Analysis* (FMEA, [7]), which was developed to investigate potential failures in systems. The results of its analytic methods provide a solid foundation for our formulation of MBT requirements.

The contribution of this paper is twofold:

1) *Failure analysis*: We analyze which properties failures in adaptive systems can encompass and which scenarios may occur by performing a FMEA-based investigation process.
2) *Requirements identification*: We derive requirements for test models based on the discovered failure scenarios. The result is a reasonable foundation for test research concerning adaptive systems.

The remainder of this paper is structured as follows: We start with related work in Section II. In Section III, we perform our FMEA-based investigation for SAS and in Section IV, we state the resulting modeling requirements. We finish in Section V and outline future work.

II. RELATED WORK

In this section, we present related approaches concerning testing adaptive systems from the perspectives of different research directions. On the one hand, *Self-adaptive Software* (SAS) [3] and *Models@run-time* (MRT) communities have to be considered as sources of adaptation concepts; on the other hand *Dynamic Software Product Lines* (DSPLs) [8] are an intersecting approach based on means like variability

 ISBN: 978-1-61208-274-5

management and features. Furthermore, *context-adaptivity* has been a long-term research field. In all these directions, several testing methods were established. In the following, we concentrate on approaches with models which can help to build up a reasonable information base for testing and on those which propose fitting coverage concepts.

A conceptional discussion on challenges concerning verification and validation of SAS was done in [3]. The authors proposed to focus on adaptive requirement engineering and runtime validation to assure adaptive software's quality. However, they also constructed an abstract model of adaptive software's states consisting of an inner system state plus a mode or phase. The latter ones describe in which variation/adaptation a system works. Each transition concerning either mode or state changes the overall configuration of the system and hast to maintain certain local or global properties. It is also discussed that a steady model as behavioral specification is insufficient such that the configurations have to conform to a dynamic selection of associated models. While these proposals are general enough to abstract from specific self-adaptive systems, several problems remain. Due to the enormous complexity in the behavioral space of adaptive software, an exact limitation of possible transitions to such which are correct and relevant for testing is a very hard task. In consequence, a much more expressive and usable model should be applied in test modeling.

Another approach by Munoz and Baudry is presented in [9]. The authors formalize context and variant models and generate sequences of context instances by using Artificial Shake Table Testing (ASTT). Thus, an adequate set of dynamic environment changes can be simulated. The case study's adaptation is designed by so-called policies, which also serve as an oracle (i.e., a mapping between inputs and outputs) in testing. Hence, with this approach testers are able to validate the correctness of the adaptation decision and to produce a sequence of re-configurations. While being a reasonable and basic approach to testing adaptive software, it lacks the means to deal with the discussed interaction challenge due to its exclusive consideration of environment changes.

An advanced research framework for adaptive software is provided by the DiVA project. DiVA had impact on both SAS and DSPL research. It also includes a methodology for testing [10][11]. DiVA's validation process is split into two phases: (1) The early validation is based on design time models (adaptation logic and context model) and executed as a simulation. A main focus in DiVA's test method is to generate reasonable context instances and associate "partial" solutions (using a test oracle), which can be used to find a set of valid configurations. The following coverage criteria are named: Simple (test each value of a variable), pair-wise (test each two-wise combination of variable values), dependency-based (reduce effort through constraints on variable values) and compound (composition of all). (2) Additionally, an operational validation method is proposed that also deals with context changes/transitions. Therefore, DiVA uses Multi-dimensional Covering Arrays (MDCA) including a temporal dimension. These describe multiple context instances that are scheduled as test sequences and provide a means for the definition of coverage criteria on sequences of adaptations. There are also fitness functions that help to minimize the test cases while sustaining a good coverage. While DiVA contributes many ideas for testing, it still does not consider interactions with the application's control flow.

Besides approaches from the DSPL and SAS research field, other work focuses on context-awareness and test data generation from context models [12][13]. For instance, Wang et al. construct in [13] control flow graphs of context changes and associate them by using point of high impact (Context-Aware Program Points, CAPPS) with the core control flow. They also provide three context-adequacy criteria. However, the proposed models of this approach are not extensive enough to express the behavior such that additional test coding is required.

III. Failure Analysis

In this section, we analyze relevant failure characteristics and scenarios. For this purpose, we use *Failure Mode and Effects Analysis* (FMEA) [7]. FMEA is used in engineering of safety-critical systems to find relevant failure sources. The method was first applied for electrical and mechanical systems and later extended for the usage in software engineering [14][15]. Based on these experiences, our analysis is separated into three steps:

1) identification of SAS-specific failure dimensions and properties (presented as *Failure Domain Model*)
2) investigation of SAS-specific failure scenarios
3) visualization of error propagation among the found scenarios as *Fault Dependency Graph*

Step (3) is not an actual part of FMEA, but usually a *Fault Tree analysis* (FTA) [16] is performed to visualize the scenarios' dependencies. As our system runs a control *loop*, we customize the analysis process in this step by constructing a fault dependency graph instead.

A. A Common Process of Self-adaptation: MAPE-K

Before starting the analysis, a level of detail has to be specified to have a fix abstraction perspective on SAS and a well-defined system boundary. FMEA is designed to be run against an existing architecture, which we cannot assume to be widely similar in all existing or future developed SAS. Hence, we leave the strict understanding of FMEA by analyzing the MAPE-K process as common concept of minimal necessary data flows with the means of this investigation method. As seen in the previous section, there are several intersecting research directions coping with self-adaptivity. They have in common that the process of information gathering and utilization can be described in a known schema, mostly referred to as the MAPE-K (Monitor/Analyze/Plan/Execute–Knowledge) [1] control loop.

As illustrated in Figure 1, this process consists of four tasks to be fulfilled by any self-adaptive system framework. The system *monitors* a certain data source such as a *sensor*. The captured information is then forwarded to the *analysis* part where the system reasons about the necessity of adaptation. After these first two process tasks the system is able to determine *if* an adaption should be performed.

In the subsequent *plan* section, an adaptation plan is generated and later applied in the following *execute* task. *Effectors* may also manipulate external entities. The control

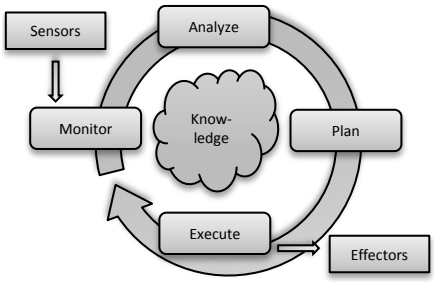


Fig. 1. MAPE-K control loop (cf. [1]).

cycle is re-run from this point periodically such that a self-adaptive system always has an optimal state (according to the supposed utility or goal function, and available knowledge) to fulfill its task. Analysis, planning, and execution interact through data organized as a central *knowledge* model.

MAPE-K encompasses the adaptation according to environment changes such as context, user input and system utilization. The sources of processed information can be abstracted by leaving out the concrete objects sensors and effectors work on. Hence, we use MAPE-K as common view point on SAS architectures.

B. Step 1) Failure Domain Model (FDM)

In this section, we provide a set of properties that failures in adaptive systems can comprise. As there are several classes of adaptive software [17], we cannot assume that each property is reasonable in every concrete system. Furthermore, we exclude failures, which originate in sensors and effectors to create a fixed boundary around the software's scope.

Each of an SAS' components provides a service (i.e., a perceivable behavior) through its interface. In the system's specification an expected behavior is defined. According to [4], a *failure* is an event of service deviation from this expectation. An *error* is the inconsistent part of the total system state (internal state plus perceivable external state) which lead to this failure. The cause of an error is a *fault*. Error propagation occurs if a failure causes a fault in another component.

The result is the fault-error-failure causal chain as presented in the FDM in Figure 2. Each concept has up to three variable dimensions. Although, in [4] several dimensions are listed, here we limit our investigation on those which are relevant to a development-independent tester who is not in charge of repairing the system. In consequence, concerning faults, the only relevant property is their *persistence* which may either be *permanent* or *transient*. For instance, intent or objective play no role when testers uncover and report defects. While persistence is a rather general dimension, this classification has an extended significance due to the cyclic nature of SAS. The source of a fault can be in one of the control loop's tasks and, additionally, originate in the system's knowledge. The latter case produces a cyclic failure propagation: If a failure manifests in the system's knowledge model, it may have harmful influence on future decisions such that the failure becomes a fault in following cycles.

For errors we propose the dimensions type and localization. Types may be one of three: The caused error either manifests

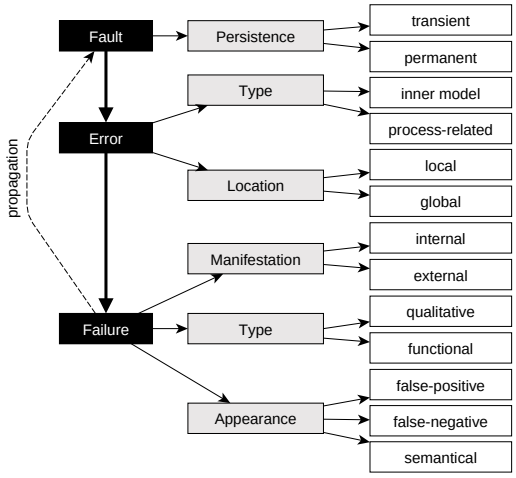


Fig. 2. Failure domain model.

as an inconsistency in (1) the knowledge representation of the perceived environment does not reflect its physical nature, (2) the model itself (e.g., it violates the model's constraints), or (3) as incorrect intermediate state in the computation process (process-related). Furthermore, errors can localize either locally or globally in the potentially distributed SAS.

Concerning failures, [4] distinguishes between content and timing (early/late) correctness. In contrast, an SAS' goal definition may aim on other non-functional properties like energy-usage. Thus, we alter the original distinction to qualitative (the system performs non-optimal) or functional (incorrect system behavior). Furthermore, failures in SAS can either manifest internally or externally (e.g., by using the effectors). The last dimension comprises the event-driven nature of the system. The sensed and analyzed information may lead to un-intentional (false-positive), missed (false-negative), or semantically wrong adaptation attempts.

This FDM is a valuable source when classifying failures in concrete adaptive systems. It describes abstract properties of failures that can be instanciated for real world systems. Verdicts (the classification of test results, for instance `Pass`, `Fail` or `Inconclusive`) can be parameterized by these information.

C. Step 2) Failure Scenarios

After defining all dimensions of failures, it is required to identify scenarios of failure occurrence in adaptive software. We cannot give a general method of priorization, as usually done in FMEA. For testing one of these scenarios, this strongly depends on domain specific conditions. As only general valuation standard, *criticality* can serve. In this case, according to [17], each adaptation operation can be either harmless, mission-critical, or safety-critical.

We decompose the MAPE-K control loop in five components: `Monitor`, `Analyzer`, `Executor`, `Planner` and `Scheduler`. The latter two are separated to enable the consideration of *interaction* between adaptation and running processes. After the `Analyzer` found *that* the system has to be adapted, the `Planner` decides *how* the adaptation is

processed. The Scheduler has the task to fill an Action queue (however it may be implemented in concrete systems) by arranging system process actions with adaptation actions. An adaptive system designer has to be aware of how he maintains consistency either through an actual implemented scheduler component or a transaction-like behavior. This issue also breaks the straight MAPE-K data flow because a scheduler requires information about the system actions (retrieved from the Executor) and composes them with adaptation intents.

All components are considered as black boxes. Components are connected by data flow edges (black arrows). Additionally, the process contains Sensors, Effectors and the central knowledge Models. The latter one contains information about the system structure adaptation logic and further knowledge relevant for adaptation decisions. Sensors and Effectors communicate with the external world (e.g., other systems or the physical reality).

Based on this structure, we list our found failure scenarios in Table I. It comprises *Failure Identifier* (FID), *Component Identifier* (CID), Fault, Error, Failure, and a *Propagation* column. All FIDs can be found in the architecture visualization in Figure 3 as well. In the following, each scenario is described in detail.

SENS: The first scenario comprises test input received from the Sensors and misinterpreted by the Monitor component (i.e., it does not produce corresponding events or produces events without being indicated by sensor inputs). For instance, a context reasoner could output an physically causeless temperature change due to a poorly designed interference rule.

TRIG: An event produced by Monitor does not or unintentionally lead to a corresponding adaptation initiation or to a wrong one. Unintended initiations let the situation *appear* (cf. appearance dimension of failures in the FDM, Figure 2) as if an adaptation is necessary. TRIG may also be caused by a propagated failure from SENS or EVENT in integration testing. Here an example is a wrong implemented adaptation rule condition.

PRE: Though the set of operated Models contains information which (according to the adaptation logic) either leads to a specific adaptation or prohibits one, the Analyzer decides differently. At this point, potential failures in the adaptation logic itself have to be considered—either due to a wrong specification or adaptation (cf. POST scenario below). For instance, consider a recorded sound level which is taken into account while reasoning about rising a sound output of the system itself. If the recorded information is erroneous, non-intended adaptions may be initiated. Such failures can also be observed while testing the Analyzer.

Both TRIG and PRE scenarios may interact, because we did not decompose the analyzer in more detailed components. Hence, these scenarios have to be tested together as both data sources are required for each test case and just a probabilistic estimation can be stated which one is actually defective.

ADAPT: The adaptation initiation may again be of false-positive, wrong or missing appearance. Like TRIG and PRE, this scenario relies on the assumption that the adaptation logic is correct. Such an scenario may occur if the adaptation reasoning mechanism misses to execute a rule's adaptation directive.

PLAN: The Analyzer determines *if* an adaptation is required but *not how* to perform it. This task is operated in the planning phase. A Planner reasons over the variability and the current system state. Its output have to be a correct adaptation plan that can be applied in the system and leads to a consistent state. The PLAN scenario encompasses that the compiled plan is incorrect, e.g., its order.

SCHED: Reconfiguration actions potentially interact with the system's control flow. Such problems arise because variability cannot be completely orthogonal to the system's inner behavior. For instance, if the system is a database, there could be a potential conflict when adapting to a situation where a speedup is required by deactivating the transaction feature in exactly the same time period when running a transaction. We get an active SCHED scenario if the compiled action sequence of the Scheduler is inconsistent.

The Executor is a complex interpretation engine that produces multiple outputs and thus, has multiple potential failure scenarios. All Executor-related scenarios may also be the outcome of a propagated SCHED failure.

RECONF: The reconfiguration may run into a failure itself. If any reconfiguration mechanism fails without being recognized, the actual system structure is out of synchronization with its model representation. Here failures can be constituted that can be of both types: functional or qualitative (cf. FDM).

POST: On the other hand, the Model's part that represents the reconfigured systems may be inconsistent after the execution because a model manipulation was performed erroneously by the the Executor. For, instance if for further adaptation decisions knowledge about past ones is required, a missed recording causes problems. POST can have complex consequences because the manipulated model is assumed to be correct in PRE and PLAN. Hence, POST may propagate faults to these two scenarios. Additionally, this scenario can also be a "starting point" failure because its semantics also resemble a defective Model at the system's start-up.

EFFECT: Another output of the Executor can be actions that have to be run in external systems by physical Effectors. If actions are not generated correctly and forwarded to the effectors (e.g., due to corrupt drivers), the representation of these externals lose synchronization with potential internal model representations. As the Sensors may perceive data from the manipulated system, we produce a possible propagation of this failure to the SENS scenario.

EVENT: The last failure scenario is related to events that are produced in the system (e.g., user interactions) and are propagated to the Analyzer component. In this case, the generated events can be erroneous.

D. Step 3) Failure Dependency Graph

As final FMEA artifact we construct a failure dependency graph as depicted in Figure 4. The visualization illustrates the potential cyclic failure propagation through inner system events, model manipulation, or physical Effectors/Sensors correlations (the latter one is visualized by the dashed edge). Furthermore the PRE and TRIG scenarios may influence each other in both directions, which makes them hard to test in isolation.

 ISBN: 978-1-61208-274-5

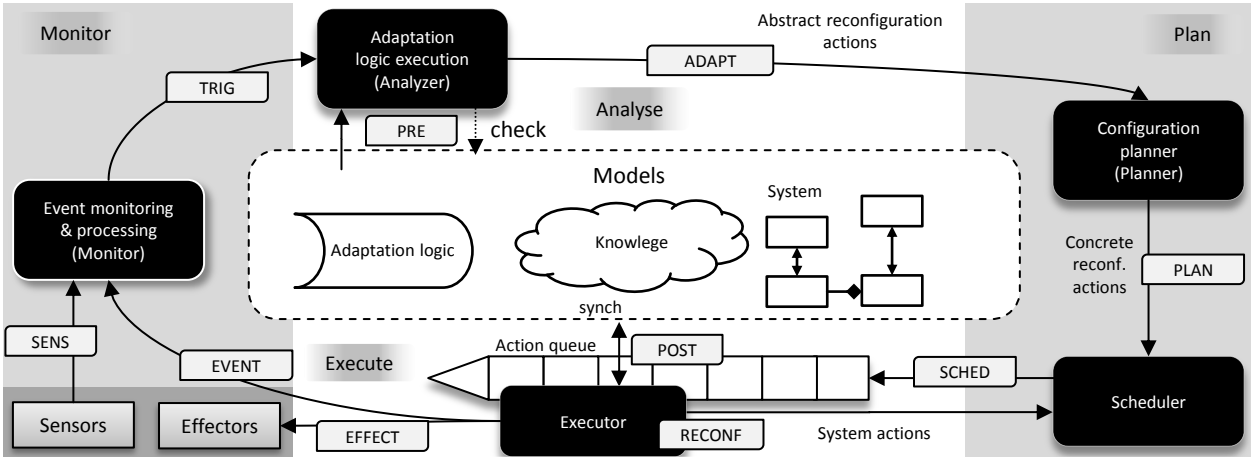


Fig. 3. Conceptional architecture of an adaptive system.

TABLE I. ADAPTIVE SOFTWARE'S FAILURE SCENARIOS.

FID	CID	Fault	Error	Failure	Propagation
SENS	Monitor	corrupt sensor interpreter	misinterpreted sensor data	no/wrong/unintended event	TRIG
TRIG	Analyzer	corrupt event interpreter	misinterpreted event	no/wrong/unintended adapt.	PRE—ADAPT
PRE	Analyzer	corrupt model interpreter	misinterpreted model	no/wrong/unintended adapt.	TRIG—ADAPT
ADAPT	Analyzer	corrupt reasoning	wrong adaptation derived	no/wrong/unintended adapt.	PLAN
PLAN	Planer	corrupt planner	inconsistent planning	inconsistent plan	SCHED
SCHED	Scheduler	corrupt scheduler	inconsistent scheduling	wrong order of actions	POST—EVENT—EFFECT—RECONF
POST	Executor	corrupt model manipulator	corrupt model construction	model inconsistent	PRE—PLAN
RECONF	Executor	corrupt configurator	reconfiguration fails	configuration↔model unsynch	–
EVENT	Monitor	corrupt event producer	wrong event production	no/wrong/unintended event	TRIG
EFFECT	Executor	corrupt forwarding	processing wrong effector operations	model↔externals unsynched	(SENS)

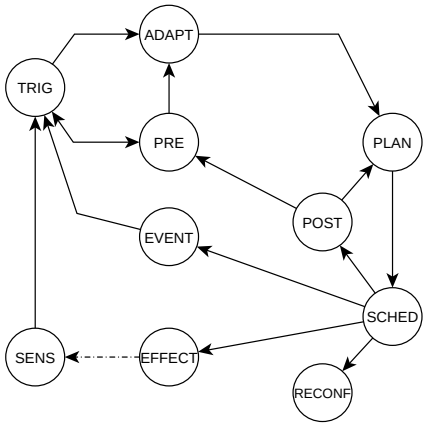


Fig. 4. Interdependencies of failure scenarios.

IV. REQUIREMENTS TO MODELS FOR SAS TESTING

All following requirements for adaptation correctness test methods are based on one or more of the presented failure scenarios. In the following, we list these mapped requirements and give each one a name for later reference. The requirements are formulated as *assurance tasks* that have to be fulfilled by employing MBT methods.

1) **Correct sensor interpretation:** Assure that the sensor data is correctly interpreted and transformed into system events. Potential sensor data has to be specified together with context identifications.(↦SENS)
2) **Correct adaptation initiation:** Assure that events initiate the correct adaptation if all preconditions in the model hold. Events, conditions, and adaptation decisions (goals) have to be associated in the models.(↦TRIG/PRE/ADAPT)
3) **Correct adaptation planning:** Assure that the generated adaptation plan is consistent w.r.t. target configuration and action order. Build a model to map adaptation goals to possible plans.(↦PLAN)
4) **Consistent interaction between adaptation and system behavior:** Assure that the generated adaptation plan is correctly scheduled with the systems' control flow. A model is required to define which adaptation is allowed in which state of application control.(↦SCHED)
5) **Consistent adaptation execution:** Assure that (1) the generated adaptation schedule is applied to system structure and (2) the synchronization between system and models is consistent after adaptation. Thus, we need a set of assertions to be checked after the adaptation execution.(↦POST/RECONF)
6) **Correct system behavior:** Assure that the system correctly commits events or actions to the effectors. As in the previous requirement, here we need to specify events to be observed in the system when running any operation.(↦EVENT/EFFECT)

 ISBN: 978-1-61208-274-5

Additionally, in testing, coverage criteria are required to restrict the combinatorial search space of the system under test and, nevertheless, have a reasonable and meaningful test result. For less-complex systems, many criteria for test coverage were found. Mostly, they refer to a graph representation like a state machine. Known criteria are statement, branch or path coverage. However, as we have seen, there is a complex set of requirements and aspects to be tested in the context of SAS. In consequence, we have to use multiple models which are more expressive then state machines (as assumed in the mentioned coverage criteria) to represent all testable aspects. In consequence, the known criteria cannot be applied directly. Hence, the last requirement is to find a set of proper coverage criteria for adaptation mechanisms which can be composed:

7. **Adaptive coverage criteria:** Find constructive coverage criteria metamodels/languages to describe *which*, *when* (in relation to system behavior), and *in which order* adaptation scenarios have to be tested and analytic coverage criteria to measure how adequate a test suite is.

V. Conclusion and Future Work

In this paper, we applied a customized Failure Mode and Effects Analysis (FMEA) to a conceptional self-adaptive software system based on the minimal structural assumptions of MAPE-K. We derived a failure domain model to provide a system in which faults, errors and failures can be classified. Subsequently, we derived ten distinct failure scenarios that occur in the process of adaptation. By building a fault dependency graph we visualized potential cyclic propagation of failures in such systems. In consequence, six *founded* modeling requirements were stated that all can be mapped to one or more of the described failure scenarios. A seventh requirement is established by the coverage question. Based on these foundations a systematic analysis of SAS is possible comprising failure properties, occurrence, and propagation. A well-designed MBT framework is comprehensive if all presented requirements are fulfilled and the all respective assurances are considered.

For further investigation, it is necessary to instantiate the found requirements for a real-world adaptive software infrastructure. If we can map this implementation to several adaptivity frameworks and express the majority of necessary test cases, our approach can be attested substantial and generic. Despite our work on mobile software testing, we recently started several research projects coping with adaptivity, namely SMAGS[18] and VICCI[19]. SMAGS (Smart Application Grids) proposes a role-based architecture, which is able of adaptive composition. VICCI searches for approaches to build adaptive Cyber-physical Systems (CPS) like Smart Homes or robots to improve our daily live in an intelligent manner. Both projects are possible test targets to our MBT strategy. Furthermore, from our experience in software testing, we should thereby sustain usability of used modeling concepts despite the complexity of the process. Especially in adaptive software this task is crucial due to the potentially huge complexity and variability in configurability and behavior.

Acknowledgments

This research has received funding within the project #100084131 by the European Social Fund (ESF) and the German Federal State of Saxony, by Deutsche Forschungsgemeinschaft (DFG) within CRC 912 (HAEC) as well as T-Systems Multimedia Solutions GmbH.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[2] T. M. King, D. Babich, J. Alava, P. J. Clarke, and R. Stevens, "Towards self-testing in autonomic computing systems," in Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems, ser. ISADS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 51–58.

[3] B. H. C. Cheng, D. Lemos, H. Giese, P. Inverardi, and J. M. et al., "Software engineering for self-adaptive systems: A research roadmap," in Dagstuhl Seminar 08031 on Software Engineering for Self-Adaptive Systems, 2008.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," Dependable and Secure Computing, IEEE Transactions on, vol. 1, no. 1, 2004, pp. 11–33.

[5] M. Utting, Practical model-based testing: a tools approach. Morgan Kaufmann, 2007.

[6] G. Püschel, R. Seiger, and T. Schlegel, "Test Modeling for Context-aware Ubiquitous Applications with Feature Petri Nets," in Modiquitous workshop, 2012.

[7] H. E. Roland and B. Moriarty, System Safety Egnineering and Managemnent, 2nd edn. John Wiley & Sons, Chichester, 1990, ch. Failure Mode and Effect Analysis.

[8] S. Hallsteinsen, M. Hichey, S. Park, and K. Schmid, "Dynamic software product lines," IEEE Computer, 2008, pp. 93–95.

[9] F. Munoz and B. Baudry, "Artificial table testing dynamically adaptive systems," 2009.

[10] V. Dehlen and A. Solberg, "DiVA methodology (DiVA deliverable D2.3)," 2010.

[11] A. Maaß, D. Beucho, and A. Solberg, "Adaptation model and validation framework final version (DiVA deliverable D4.3)," 2010.

[12] T. Tse, S. Yau, W. Chan, H. Lu, and T. Chen, "Testing context-sensitive middleware-based software applications," 28th Annual International Computer Software and Applications Conference, 2004, pp. 458–466.

[13] Z. Wang, S. Elbaum, and D. S. Rosenblum, "Automated generation of context-aware tests," 29th International Conference on Software Engineering (ICSE), 2007, pp. 406–415.

[14] H. Sozer, B. Tekinerdogan, and M. Aksit, Archtitecting dependable systems IV. Springer, 2007, ch. Extending failure models and effects analysis approach for reliability analysis at the software architecture design level.

[15] B. Tekinerdogan, H. Sozer, and M. Aksit, "Software architecture reliability analysis using failure scenarios," Journal of Systems and Software, vol. 81 (4), 2008, pp. 558–575.

[16] J. Dugan, Handbook on Software Reliability Engineering. McGraw-Hill, New York, 1996, ch. 15. Software System Analysis Using Fault Trees, pp. 615–659.

[17] J. Andersson, R. Lemos, S. Malek, and D. Weyns, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Modeling Dimensions of Self-Adaptive Software Systems, pp. 27–47.

[18] C. Piechnick, S. Richly, S. Götz, C. Wilke, and U. Aßmann, "Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids," in Proceedings of ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-adaptive Systems and Applications, 2012, pp. 93–102.

[19] D. B. Martin Franke and T. Schlegel, "Towards a flexible control center for cyber-physical systems," in Modiquitous workshop, 2012.

 ISBN: 978-1-61208-274-5

StaCo: Stackelberg-based Coverage Approach in Robotic Swarms

Kateřina Staňková, Bijan Ranjbar-Sahraei, Gerhard Weiss, Karl Tuyls
Department of Knowledge Engineering
Maastricht University
Email: {k.stankova,b.ranjbarsahraei,gerhard.weiss,k.tuyls}@maastrichtuniversity.nl

***Abstract*—The Lloyd algorithm is a key concept in multi-robot Voronoi coverage applications. Its advantages are its simplicity of implementation and asymptotic convergence to the robots' optimal position. However, the speed of this convergence cannot be guaranteed and therefore reaching the optimal position may be very slow. Moreover, in order to ensure the convergence, the Hessian of the corresponding cost function has to be positive definite all the time. Validation of this condition is mostly impossible and, as a consequence, for some problems the standard approach fails and leads to a non-optimal positioning. In such situations more advanced optimization tools have to be adopted. This paper introduces Stackelberg games as such a tool. The key assumption is that at least one robot can predict short-term behavior of other robots. We introduce the Stackelberg games, apply them to the multi-robot coverage problem, and show both theoretically and by means of case studies how the Stackelberg-based coverage approach outperforms the standard Lloyd algorithm.**

Keywords*—*Swarm robots; Coverage control; Lloyd algorithm; Game theory; Stackelberg games

I. Introduction

In recent years many researchers in robotics, control, and computer science have focused on swarm robotics and have developed solutions of fundamental *swarm robotic* problems (see [1] for solving flocking control problem, [2] for aggregation, [3] for multi-robot coverage, and [4] for formation). However, most of the proposed solution methods encounter difficulties in real-world applications, such as finding only sub-optimal solutions and the inability of the algorithms to account for non-convex environments. Subsequently, despite the wide range of existing works in the domain of multi-robot coverage [3], [5]–[9], there are still only very few in-field deployments, due to a wide gap between the theory of multi-robot coverage systems and the practice.

The Stackelberg Coverage (StaCo) approach proposed in this paper addresses the deficiencies of the existing works in multi-robot coverage, by adding one or more relatively advanced robots, called leaders, to the swarm. In other words, we assume a priori a heterogeneous robotic swarm, similar to that shown in Figure 1. In this figure, two intelligent robots act as the leaders, which can perceive the environment globally, and a large swarm of simple robots following simple local rules. The main advantage of such a heterogeneous approach is preserving the simplicity of the major population of the robotic swarm, while a small group of robots can predict behavior of the others and act so that the desired behavior is achieved faster and with a higher precision. The main building block of our approach is the so-called *Stackelberg game theory* [10], [11], which belongs to the more general noncooperative game theory [10], [12]. Game theory has been successfully applied in various fields; its known applications in the robotic field relate to pursuit-evasion and search problems [13], [14]. However, application of the Stackelberg games in the multi-robot coverage is new.

The remainder of the paper is structured as follows: A motivation example of classical coverage limitations will be presented in Section II. In Section III we will briefly review the game-theoretic preliminaries and introduce Stackelberg games. In Section IV the Voronoi-based coverage problem will be defined as a Stackelberg game and its properties will be discussed. The simulation setup and the results of applying the proposed approach will be presented in Section V. In Section VI we will discuss the advantages of the StaCo approach and give concluding remarks.

II. Motivation

A motivaton example, which illustrates the limitations of classical approaches in mutli-robot coverage, is shown in Figure 2. The group of robots, initiated in the position depicted in Figure 2a, moves based on the standard coverage approach suggested in [3]. With this approach, the robots are driven to the final configuration shown in Figure 2b. However, this configuration is sub-optimal (The globally optimal solution will be found adopting the StaCo approach proposed in this paper in Section V, Figure 7c). The problem of being enmeshed

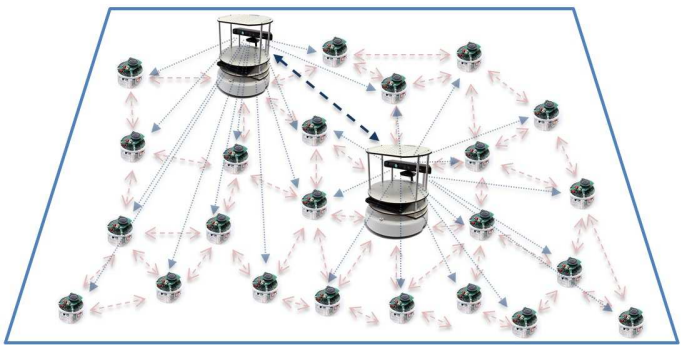


Figure 1. A heterogeneous robotic swarm with 2 leading robots and 34 following robots. The followers can collect information only from their neighbors, while the leaders are capable of collecting information from the entire robotic swarm. The leaders may be able to predict possible future reactions of the followers and to enforce their own decisions on the followers.

 ISBN: 978-1-61208-274-5

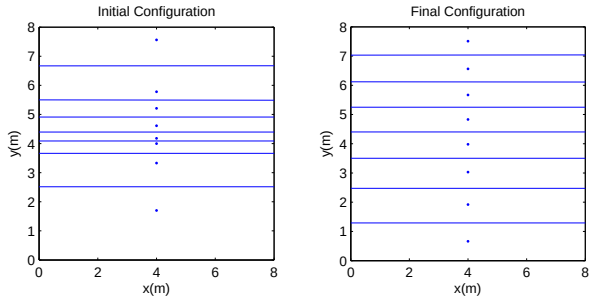


Figure 2. Example of the problem in which the standard coverage approach leads to only a locally optimal but not the globally optimal configuration (dots denote robot locations and lines denote boundaries of the Voronoi regions): (a) initial configuration, (b) final configuration achieved by approach suggested in [3]; this final configuration is suboptimal

in a local optimum can also be seen in non-convex environments (e.g. in presence of obstacles). With the motivation to avoid such complications and to speed up the procedure of finding the global optimum, we introduce the StaCo approach. Adopting this approach, the majority of the swarm consists of simple robots following local rules introduced in [3], while one or two more advanced robots (leaders) improve the system performance by taking different actions, taking the decisions of the others into account. Consequently, the decentralized behavior of the swarm and the simplicity of most robots is preserved, while overall system performance is significantly improved.

III. Basics of Stackelberg Games

Let us explain basics of Stackelberg games by the following static example.

Example III.1. *(Two-player static game) Let two players L and F have decision variables $u_L \in \mathbb{R}$ and $u_F \in \mathbb{R}$, respectively. Let functions $J_L : \mathbb{R}^2 \to \mathbb{R}$ and $J_F : \mathbb{R}^2 \to \mathbb{R}$ be smooth and strictly convex on $\mathbb{R}^2$. Player L chooses $u_L \in \mathbb{R}$ in order to minimize her cost $J_L(u_L, u_F)$, while player F minimizes $J_F(u_L, u_F)$ by choosing $u_F \in \mathbb{R}$. Illustration of this situation is given in Figure 3, where level curves (contours) of J_L and J_F are depicted in (u_L, u_F)-plane. If there is no hierarchy between L and F (i.e., if they either act simultaneously or if they do not know how the other player acts), the Nash equilibrium applies [15]. In such a situation, L and F would pick $u_L^{(N)}$ and $u_F^{(N)}$, respectively, where $u_L^{(N)} = \arg\min_{u_L} J_L\big(u_L, u_F^{(N)}\big)$, $u_F^{(N)} = \arg\min_{u_F} J_F\big(u_L^{(N)}, u_F\big)$. The outcome of the game would be $J_L\big(u_L^{(N)}, u_F^{(N)}\big)$ and $J_F\big(u_L^{(N)}, u_F^{(N)}\big)$ for L and F, respectively (i.e., the values of J_L and J_F evaluated at point $N = \big(u_L^{(N)}, u_F^{(N)}\big)$, which is the* Nash solution (equilibrium) *of the game. Note that in Figure 3, point $N = \big(u_L^{(N)}, u_F^{(N)}\big)$ lies on the intersection of the curves $R_L(u_F)$ and $R_F(u_L)$ defined by $\frac{\partial J_L}{\partial u_L} = 0$ (bold dotted curve) and $\frac{\partial J_F}{\partial u_F} = 0$ (bold dashed curve), respectively.*

Let us now consider a different situation: Player L (in this new situation referred to as leader*) knows $R_F(u_L)$ (bold dashed curve) in advance and can act first. In such a situation it is better for the leader to choose $u_L^{(S)} = \arg\min_{u_L} J_L\big(u_L, R_F(u_L)\big)$ instead of $u_L^{(N)}$. Subsequently, follower F chooses $u_F^{(S)}$ (there is no other u_F for which $J_F\big(u_L^{(S)}, u_F\big) < J_F\big(u_L^{(S)}, u_F^{(S)}\big)$). Point $S = \big(u_L^{(S)}, u_F^{(S)}\big)$ is then the* Stackelberg solution (equilibrium) *of the game and $J_L\big(u_L^{(S)}, u_F^{(S)}\big)$, $J_F\big(u_L^{(S)}, u_F^{(S)}\big)$ are Stackelberg outcomes of this game for the leader and the follower, respectively [10], [11].*

We will now generalize the example. Let us state first the assumptions that we raise on the cost functions and decision spaces in the static game:

- (A1) Let Γ_L and Γ_F be convex compact sets, referred to as decision spaces for the leader and follower, respectively.
- (A2) Let $J_L : \Gamma_L \times \Gamma_F \to \Gamma_L$ and $J_F : \Gamma_L \times \Gamma_F \to \Gamma_F$ be strictly convex smooth functions on $\Gamma_L \times \Gamma_F$, referred to as costs for the leader and follower, respectively.

Imposing assumptions (A1) and (A2), we provide following definitions:

Definition III.2. *(Optimal response set in the static game) Under assumptions (A1) and (A2), the set $R(u_L) \subset \Gamma_F$ defined for each strategy $u_L \in \Gamma_L$ of L by $R(u_L) = \{\xi \in \Gamma_F : J_F\big(u_L, \xi\big) \leq J_F\big(u_L, u_F\big), \forall u_F \in \Gamma_F\}$ is the* optimal response set *for F.*

Definition III.3. *(Stackelberg strategy in the static game) Under assumptions (A1) and (A2) and with $R(u_L)$ unique for each $u_L \in \Gamma_L$, strategy $u_L^{(S)} \in \Gamma_L$ is called a Stackelberg equilibrium strategy for L if $J_L\big(u_L^{(S)}, R(u_L^{(S)})\big) = \min_{u_L \in \Gamma_L} J_L\big(u_L, R(u_L)\big)$.*

The existence and uniqueness of Stackelberg strategy is discussed in following lemma:

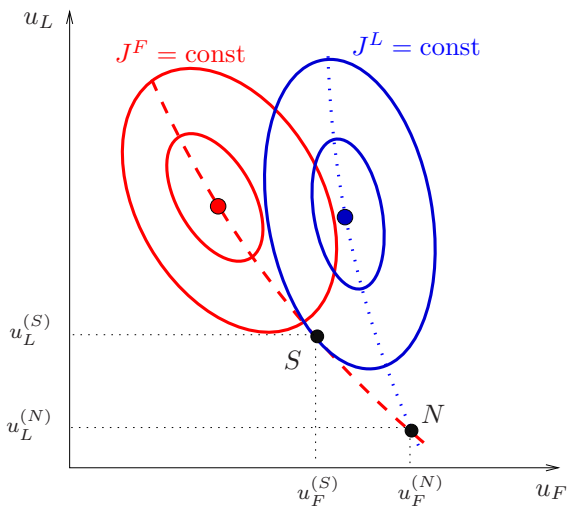


Figure 3. Illustration of the difference between Stackelberg (S) and Nash (N) equilibrium solutions. When compared to the Nash equilibrium, under the same conditions of the game the Stackelberg equilibrium never leads to higher costs for the leader (provided that they both exist). Moreover, there are situations in which the Stackelberg equilibrium concept might be more profitable for the follower as well, as this figure illustrates.

 ISBN: 978-1-61208-274-5

Lemma III.4. *(Existence and uniqueness of Stackelberg strategy) Every two-person static game with leader L and follower F, where (A1) and (A2) hold, admits a unique Stackelberg strategy for the leader.*

Proof. *If Γ_L and Γ_F are convex compact sets and $J_L : \Gamma_L \times \Gamma_F \to \Gamma_L$ and $J_F : \Gamma_L \times \Gamma_F \to \Gamma_F$ are strictly convex smooth cost functions, then $R(u_L) \subset \Gamma_F$ by Definition III.2. Existence and uniqueness of the Stackelberg strategy directly follow from Definition III.3.*

To conclude this short introduction, we state the obvious property of the Stackelberg outcome.

Lemma III.5. *(Stackelberg outcome versus Nash outcome in a static game) For a two-person static game with leader L and follower F, where assumptions (A1) and (A2) hold, $J_L\big(u_L^{(S)}, u_F^{(S)}\big) \leq J_L\big(u_L^{(N)}, u_F^{(N)}\big)$.*

If the decisions of the players and the state of the system evolve in time, while each of these decisions and the state of the system influence (also future) decisions and states, we refer to the game as the *dynamic* game. Without going into too much detail, we state that theory introduced in this section for static games can be extended into the dynamic setting, in both discrete-time dynamic and continuous dynamic cases, under additional assumptions on the system dynamics. For an overview of theory of Stackelberg games with varying information each of the players might know, see [10], [11], [16], [17]. Moreover, a Stackelberg game can also be played among one leader L and multiple followers F_1, ..., F_M, where the leader, having complete information about the state, cost functions, and dynamics of the followers can impose her decision on the followers at each time step $k \in \{1, \ldots, N\}$ (resp. each time $t \in [0, T]$) in the discrete and continuous case, respectively.

IV. StaCo approach

In this section we formulate multi-robot coverage problem as a dynamic Stackelberg game with one leader and multiple followers. The approach proposed in this section will be referred to as StaCo: Stackelberg-based Coverage Approach.

Let us consider M robots (players) positioned at time $t = 0$ in convex polytope $\Omega \subset \mathbb{R}^2$. One of the players, denoted for the sake of simplicity as player 1, is the *leader*, other players, denoted by 2, ..., M, are the *followers*. Let $\mathbf{x}(t) \stackrel{\text{def}}{=} \{x_1(t), x_2(t), \ldots, x_M(t)\}$ be the configuration of the robots at time t, with $t \in [0, T]$, $\mathbf{x}(0) = \{x_1(0), x_2(0), \ldots, x_M(0)\}$ being the a priori given initial configuration of the robots and $\mathbf{x}(T) = \{x_1(T), x_2(T), \ldots, x_M(T)\}$ being their final configuration at final time T, with $x_i(t) \neq x_j(t)$ if $i \neq j$. Let $V_i(t)$ indicate the Voronoi region (cell) in which i-th robot is located at time t. For each $\mathbf{x}(t)$ the Voronoi regions are defined by the *Voronoi partition* of Ω, $\mathcal{V}(t) = \{V_1(t), \ldots, V_M(t)\}$ generated by the points $\mathbf{x}(t) = \big(x_1(t), \ldots, x_M(t)\big) : V_i(t) = \{\omega \in \Omega : \|\omega - x_i(t)\| \leq \|\omega - x_j(t)\|, \quad \forall j \neq i\}$. System dynamics (with state variable $\mathbf{x}$) are given by the following system of ordinary differential equations:

$$\dot{x}_i(t) = u_i(t), \quad i = 1, \ldots, M \tag{1}$$

where $u_i(t)$ is the control (decision) of the i-th robot at time t. The cost functions for the leader (robot 1) at time t is given by

$$C_1(t) = \sum_{i \in \{1,\ldots,M\}} \int_{V_i(t)} \|\omega - x_i(t)\|^2 \mathrm{d}\omega. \tag{2}$$

Let T be the stopping time, i.e. the minimal time such that for each $\tau > T$ the cost $C_1(\tau)$ does not change: $T = \min\{t : C_1(\tau) = C_1(T) \text{ for } \forall\ \tau > T\}$. Then the leader minimizes $C_1(T)$. The cost function for the follower $j \in \{2, \ldots, M\}$ at time t is

$$C_j(t) = \int_{V_j(t)} \|\omega - x_j(t)\|^2 \mathrm{d}\omega. \tag{3}$$

The problem of the leader (robot 1) can be then defined as

$$(P_{\text{StaCo}}) \begin{cases} \text{Find} \quad u_1^{(S)}(t) = \arg\min\limits_{u_1(t)} C_1(T), \text{ w.r.t.} \\ u_j(t) = \arg\min\limits_{u_j(t)} \int_{V_j(t)} \|\omega - x_j(t)\|^2 \mathrm{d}\omega. \\ \dot{x}_i(t) = u_i(t), \end{cases}$$

with $j = 2, \ldots, N$, $i = 1, \ldots, N$.

The solution strongly depends on the so-called information pattern, i.e., on the amount of information that each player knows and recalls over her own state, state of the others, and action made by herself and the others during the game.

Proposition IV.1. *Let at time t each player i know only state $x_i(t)$ and corresponding $V_i(t)$ and let Hessian of* (2) *be positive definite at each t. Then the so-called continuous-time Lloyd descent [3]*

$$u_i^*(t) = \kappa \left(\frac{\int_{V_i(t)} \mathbf{x}\, \mathrm{d}x_i}{\int_{V_i(t)} \mathrm{d}x_i} - x_i(t) \right), \tag{4}$$

$\kappa > 0$, asymptotically converges to minimal $C_1(T)$ for player 1 and to minimal $C_j(T)$ for $j = 2, \ldots, M$.

Proof: As shown in [3], $u_i^*(t)$ defined by (4) with respect to $\dot{z}_i(t) = u_i(t)$ converges asymptotically to the set of critical points of (2). The critical points of (2) coincide with critical points of (3). If corresponding V_i is finite, this solution is global due to positive definiteness of (2) [18]. ■

Remark IV.2. *Note that validating the positive definiteness of* (2) *is an open problem [3] and even if the convergence to the global optimum is guaranteed, in general no guarantees on the speed of this convergence exist. This leads us to the question whether there exist algorithms that perform better than the classical Lloyd algorithm if we allow the leader (robot 1) to have more information about the state and decisions of the followers.*

Proposition IV.3. *Let player 1 know $x_j(\tau)$ and $u_j(\tau)$ (for all $j \neq 1$) for $\tau \in [t, t+\Delta]$, with $\Delta > 0$, where $u_j(t)$ is defined by* (4)*. Let $u_1^{(S)}(t)$ denote the optimal control of player 1, possibly dependentent on $u_j(\tau)$, $\tau \in [t, t+\Delta]$. Let T^Δ, and $C_1^\Delta(T^\Delta)$ denote the corresponding stopping time and the final payoff for player 1 in such a situation, respectively. Then $T^\Delta \leq T$ and $C_1^\Delta(T^\Delta) \leq C_1(T)$.*

Proof: The leader's decision is not bounded by any restrictions. Setting this decision to (4) leads to $T^\Delta = T$,

$C_1^{\Delta}(T^{\Delta}) = C_1(T)$. Note that the Hessian of (2) might not be positive definite with the leader's decision defined by (4). Thus, $u_1^{*,S}(t)$ either coincides with (4) or, if this choice would lead to only sub-optimal solution, $u_1^{*,S}(t)$ differs from (4) and leads to a better outcome. This result also follows from extension of Lemma III.5 into dynamic setting with the state equation (1). ■

Giving more information to the leader almost always leads to the better outcome for the leader also in a very general setting [10], [11], while the StaCo approach never leads to the outcome worse than that reached by standard methods [3]. In the next section we will illustrate that when the classical Lloyd algorithm fails and leads to only a local optimum, the StaCo approach can find the global solution. For the case studies in the next section, the time and space are discretized and therefore the leader can choose from a limited number of decisions at each time step k.

V. Case Studies

In this section, we will study the performance of the proposed StaCo approach in comparison with the classical Voronoi-based coverage approach.

A. Simulation Setup

To simulate StaCo and compare it with the standard approach, we have developed a 2D robot simulator. This simulator is written in Java and supports simple massless robot motion. The environment Ω to be covered in all simulations is a 8 m $\times$ 8 m square and the speed of each robot is limited to 4 cm/s. The time discretization of the system is 0.4 s.

The designed simulator supports Voronoi cell computation for each robot. In each time step, firstly the locations of robots x are used to compute the Voronoi cell of each robot and subsequently the centroid of each cell is computed and used by the robots to find the gradient descent direction (4). With the StaCo approach the robot closest to the center of Ω is considered as the leader. In each time step, instead of following the gradient descent direction (4), the leader first discretizes its surrounding space into a limited number of accessible locations (in our simulations 8 points on a circle of radius 1.5 cm, with equal distances to each other). Then for moving to each of these locations, the leader predicts the possible moves of other robots, in one or two time steps, and chooses the movement, which minimizes C_1 (i.e. the best possible response to the other robots).

In order to measure the performance of the StaCo approach and to compare it with the performance of the classical coverage techniques, we introduce the *Settling Time* as the time required for the cost function (2) of the whole swarm to enter and remain within a prespecified error boundary. More precisely, we define the settling time T_s as

$$T_s \stackrel{\text{def}}{=} \min\left\{T_s \in [0, T_f] \middle| \forall t > T_s : \left|\frac{C_1(T_f) - C_1(t)}{C_1(T_f) - C_1(0)}\right| < \epsilon\right\} \quad (5)$$

where $\epsilon = 0.05$, C_1 is defined by (2), and T_f is the simulation stopping time, i.e. the minimal time such that the cost $C_1(.)$ doesn't change.

As an example of the simulation setup, Figure 4a shows an initial configuration of a robotic swarm of 8 robots. Both

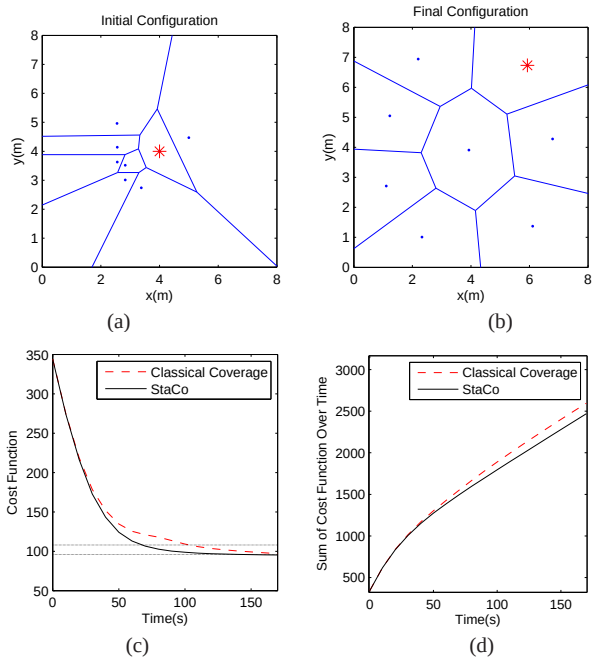


Figure 4. Comparison of performance of the proposed StaCo approach and the standard approach for a particular configuration: (a) initial configuration (b) final configuration (c) coverage cost function (d) cost function summed up over time.

the StaCo and the classical coverage approaches are applied to this configuration; with the StaCo approach the leader makes a prediction of the swarm behavior for one subsequent time step. The final configuration after 170 s is shown in Figure 4b. Clearly, both methods reach the same final configuration; however, as shown in Figure 4c, StaCo reaches the final configuration faster than the classical approach. Finally, in Figure 4d, the cost functions for both techniques are summed up over the time. This figure shows that the StaCo approach converges to the optimal configuration faster than the classical approach. The settling time of both approaches can be easily measured via the horizontal lines in Figure 4c (The upper and lower lines refer respectively to $C_1(T_s)$ and $C_1(T_f)$, which denote the 0.05 error bound). Therefore, in this particular case study, the settling time for the StaCo approach is 75 s, and for the classical coverage approach it is 105 s.

B. Effect of Swarm Size

In order to compare both techniques in a more generic way, we have applied our simulation to groups of 2 – 20 robots, 20 times for each swarm size, with random starting configurations. The convergence settling times for both techniques were accurately measured based on (5). Their statistical representation is illustrated in Figure 5. In this figure, the average value, the minimum, and the maximum of the settling time over 20 runs are plotted with respect to the swarm size. From Figure 5 we can conclude that the StaCo approach performs better compared to the classical coverage approach. For certain initial configurations both methods achieve the final configuration with the same settling time, while for the majority of possible initial configurations the StaCo approach performs better. Such behavior is observed in simulations and is also supported by the theoretical arguments in Sections III and IV.

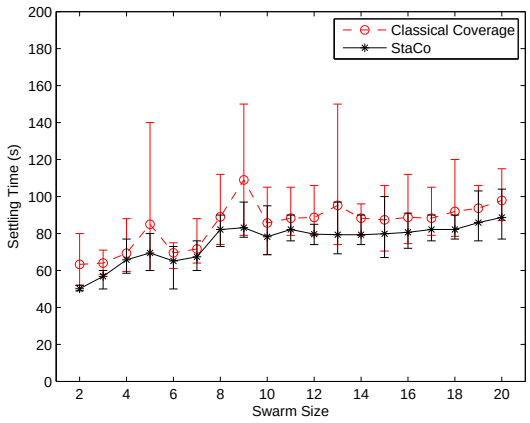


Figure 5. Comparison of the coverage settling time between the proposed StaCo approach and the classical coverage approach for robotic swarms of different sizes.

C. Effects of Leader's Speed and Prediction Horizon

Firstly, we examine the effect of the leader's speed on the performance of StaCo. Secondly, we will investigate how the number of prediction steps influences the performance.

We employ a robotic swarm with eight robots. For each initial configuration, we increase the speed of the leader from 4 cm/s up to 16 cm/s in steps of 2 cm/s, while the followers' maximum speed remains 4 cm/s. Each simulation is repeated 20 times from random initial configurations. Afterwards, the simulations are repeated with the leader's prediction horizon being increased to up to 2 subsequent time steps, with varying leader's speed.

The results presented in Figure 6 show that increase of the leader's speed and prediction horizon can improve the convergence performance of the StaCo approach.

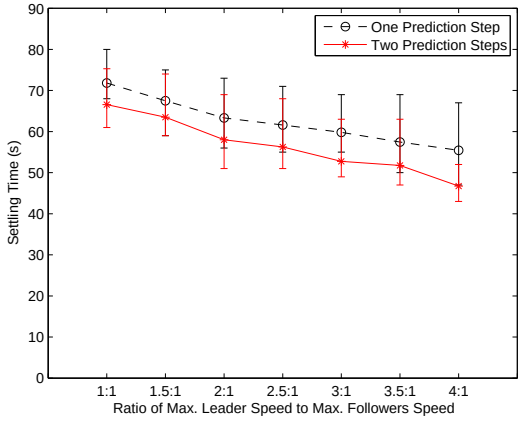


Figure 6. Coverage settling time of a robotic swarm of one leading robot and seven following robots for different leader's speeds, while the leader makes predictions for one or two future time steps.

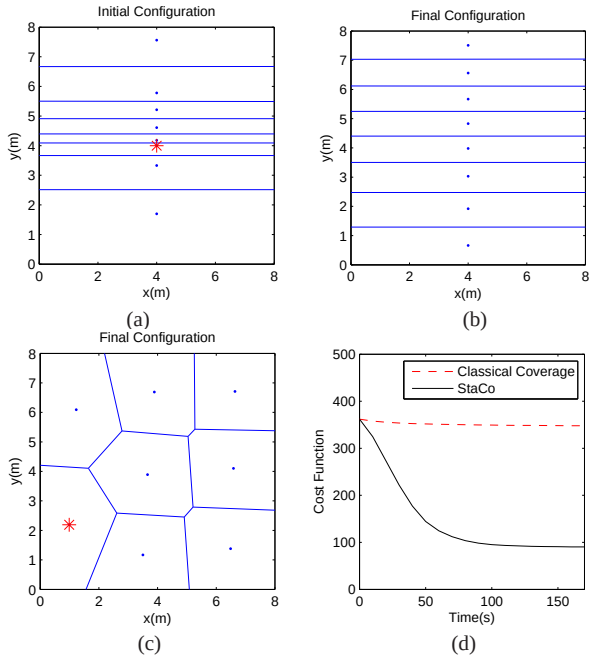


Figure 7. Comparison of coverage performance between the proposed StaCo approach and the standard coverage approach for an initial configuration close to a sub-optimal configuration: (a) initial configuration; (b) final configuration for standard coverage approach; (c) final configuration for the StaCo approach; (d) comparison of the cost functions.

D. Escaping sub-optimal configurations

In StaCo approach the leader is able to perceive global information about the position of all swarm robots. This ability may help the swarm to escape from sub-optimal configurations. A sample initial configuration, already discussed in Section II, is shown in Figure 7a.

This initial configuration is very close to a suboptimal case, which is achieved if each robot moves a bit up or down and settles in the center of its rectangular Voronoi cell. Although the classical coverage approach terminates in this local minimum immediately (see Figure 7b), it is very easy for the StaCo approach to escape from this local minimum. The final configuration achieved by the StaCo is shown in Figure 7c. Comparison of costs over the time are illustrated in Figure 7d. Clearly, the StaCo approach performs much better.

Similarly to the results depicted in Figure 7, starting from any other initial configuration close to a sub-optimal configuration, the standard coverage approach will result in this sub-optimal position. The perception capabilities of the leader in StaCo allow for finding the globally optimal configuration.

VI. Discussions and Conclusions

This article addressed the multi-robot coverage problem and presented a new approach called StaCo, which is based on the game-theoretic concept of Stackelberg games. StaCo takes advantage of the high perception capabilities of a small group of robots (leaders) among a large group of simple robots (followers) and allows for a very efficient coverage performance. No communication among the robots takes place. The leader(s) choose(s) a position in such a way that the other robots will, by optimizing their own objectives, improve the

 ISBN: 978-1-61208-274-5

overall configuration of the system. Therefore, this approach is a non-intrusive way to steer the system into a desirable direction and leads to fast and effective coverage of an environment.

StaCo always performs at least as well as the classical approach, mostly StaCo performs better. This outcome was shown both theoretically and by means of case studies. Moreover, StaCo is able to escape from sub-optimal configurations when the classical approach is doomed to fail.

A possible limitation of the StaCo approach is that currently there is no explicit form of the optimal Stackelberg solution of the game due to the complexity of the cost function of the leader; however, its derivation is a subject of our ongoing research.

StaCo opens a promising new research avenue: Using heterogenous robotic swarms for coverage in complex scenarios such as those with non-convex environments (environments with obstacles or with non-convex boundaries). As described in many existing works, accomplishing a swarm robotic mission in a non-convex environment is a difficult task. However, the authors believe that the StaCo approach can be very successful in such scenarios.

REFERENCES

[1] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[2] A. Martinoli, A. Ijspeert, and L. Gambardella, "A probabilistic model for understanding and comparing collective aggregation mechanisms," in *Proceedings of the Fifth European Conference on Artificial Life (ECAL99)*, pp. 575–584, 1999.

[3] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[4] W. Ren and N. Sorensen, "Distributed coordination architecture for multi-robot formation control," *Robotics and Autonomous Systems*, vol. 56, no. 4, pp. 324–333, 2008.

[5] Z. Butler and D. Rus, "Controlling mobile sensors for monitoring events with coverage constraints," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1568 – 1573, 2004.

[6] L. C. A. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. C. Mesquita, and G. A. S. Pereira, "Simultaneous coverage and tracking (SCAT) of moving targets with robot networks," in *Proceedings of the Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR 08)*, pp. 85–99, 2009.

[7] M. Schwager, D. Rus, and J. J. Slotine, "Decentralized, adaptive coverage control for networked robots," *International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[8] A. Breitenmoser, M. Schwager, J. Metzger, R. Siegwart, and D. Rus, "Voronoi coverage of non-convex environments with a group of networked robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4982–4989, 2010.

[9] B. Ranjbar-Sahraei, G. Weiss, and A. Nakisaee, "A multi-robot coverage approach based on stigmergic communication," in *Multiagent System Technologies*, vol. 7598 of *Lecture Notes in Computer Science*, pp. 126–138, Springer, 2012.

[10] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. Philadelphia, Pennsylvania: SIAM, 1999.

[11] K. Staňková, *On Stackelberg and Inverse Stackelberg Games & Their Applications in the Optimal Toll Design Problem, the Energy Market Liberalization Problem, and in the Theory of Incentives*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 2009.

[12] M. Osborne, *An Introduction to Game Theory*. New York: Oxford University Press, 2004.

[13] E. Raboin, D. Nau, U. Kuter, S. K. Gupta, and P. Svec, "Strategy generation in multi-agent imperfect-information pursuit games," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 947–954, 2010.

[14] Y. Meng, "A game-theory based multi-robot search approach for multiple targets," *IEEE Robotics & Automation Magazine*, vol. 5, no. 4, pp. 341–350, 2008.

[15] J. Nash, "Noncooperative games," *Annals of Mathematics*, vol. 54, pp. 286–295, 1951.

[16] K. Staňková and B. De Schutter, "Stackelberg equilibria for discrete-time dynamic games – Part I: Deterministic games," in *Proceedings of the 2011 IEEE International Conference on Networking, Sensing and Control*, (Delft, The Netherlands), pp. 249–254, Apr. 2011.

[17] K. Staňková and B. De Schutter, "Stackelberg equilibria for discrete-time dynamic games – Part II: Stochastic games with deterministic information structure," in *Proceedings of the 2011 IEEE International Conference on Networking, Sensing and Control*, (Delft, The Netherlands), pp. 255–260, Apr. 2011.

[18] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: applications and algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999.

 ISBN: 978-1-61208-274-5

EvoRoF: A Framework for On-line and On-board Evolutionary Robotics

Florian Schlachter, Patrick Alschbach and Katja Deuschl
Institute for Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
{Florian.Schlachter, Patrick.Alschbach, Katja.Deuschl}@ipvs.uni-stuttgart.de

***Abstract*—In this paper, we present an evolutionary robotics framework (EvoRoF) for on-line and off-line evolution, as well as on-board and off-board evolution for swarm and reconfigurable robotics. It enables both, the use of artificial neural networks and spiking neural networks and combines both with structural evolution of recurrent networks. It is evaluated with benchmark tests and several use cases are outlined.**

Keywords-on-line evolution; recurrent neural networks; reconfigurable robotics; evolutionary robotics.

I. INTRODUCTION

In swarm robotics, a group of autonomous robots with limited sensors and actuators performs in a cooperative way. These robots often have only limited power resources and local information. Therefore, these robots are forced to take care of power recharging and efficient task allocation to ensure the correct processing of the desired task.

In reconfigurable robotics, a group of robots is able to reconfigure or aggregate into various configurations to generate new functionalities and thus gain a high degree of versatility [1], [2]. New functionalities can arise and the robotic system adapts to different operational demands to solve advanced tasks.

While the swarm and reconfigurable robotics describe a class of robotic systems, Evolutionary Robotics is a way to obtain a desired controller by applying Darwinian mechanisms [3] to the controller of those robots. The artificial evolution of robot controllers enables a swarm or reconfigurable robots to evolve over time in order to adapt to a specific task or to survive in a dynamic environment.

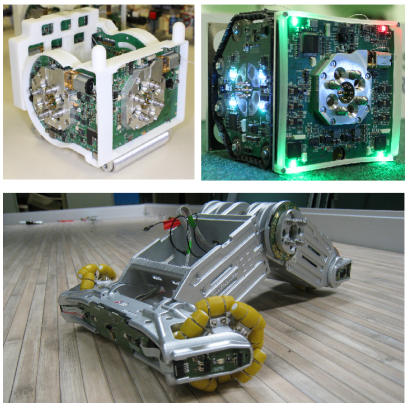


Fig. 1. The three robots developed in the Symbrion and Replicator projects. Top left: Backbone robot. Top right: Scout robot. Bottom: Active Wheel.

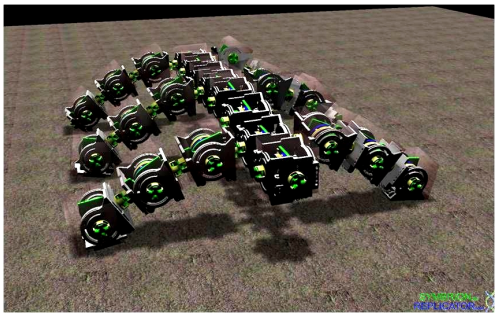


Fig. 2. An aggregated hexapod organisms of Backbone robots in the simulation to demonstrate the capabilities of the reconfigurable mobile robot platform.

Combining all three topics into one platform like in the Symbrion [4] and Replicator [5] projects, delivers a powerful robotic system for dynamic environments and unforeseen situations. Autonomous individual robots can aggregate on demand to artificial organisms with new functionaity and thus extend its operational scope. The automatic design by artificial evolution can be followed by lifelong on-line adaptation.

Thereby, the evolvability of a platform directly affects the level of adaptation and learning in robotic control. Without evolvability, a technical system is not able to change the underlying structure of control for adaptation and learning reasons. In a former paper [6], we showed the different levels of evolvability in the Symbrion and Replicator projects.

We outlined how the mechanical design has to be and the requirements of the embedded software, which we presented in [7], called Symbricator Robot API. Beside the supporting electronics, mechanics and basic software design, the platform itself extends the system by the capability of aggregation. By self-assembling, an artificial robot organism can generate new functionality in order to adapt to a changing environment or task. Figure 1 shows the heterogeneous robots in the Symbrion and Replicator projects. A detailed description can be found in [8], [9]. An aggregated multi-robot organism in simulation can be seen in Figure 2.

 ISBN: 978-1-61208-274-5

In the following we will demonstrate a framework for evolvable robot control, which enables the evolutionary design of robot controllers in individual robots as well as for cooperating robots and artificial organisms. The paper is organized as follow: Section II gives an overview about existing frameworks and related work. Section III outlines the requirements for such a framework and Section IV lines out the actual implementation of the framework, while section VI demonstrates use-cases and experiments. In Section VII, we summarize and conclude the paper.

II. Related Work

One of the earliest approaches is the Generalized Acquisition of Recurrent Links (GNARL) from Angeline et al. [10]. In this framework, the first time an algorithm is enabled to evolve the parameters and the structure of a neural network at the same time without any constraints to the topology of the network. New links are introduced with zero weight to avoid a radical change in the behaviour of a network. New neurons are introduced without any incident links, later mutations are connecting those neurons then in an appropriate manner.

The NeuroEvolution of Augmenting Topologies (NEAT), developed by Stanley et al. [11] is a generation-based framework which allows the evolution of recurrent networks from a minimal initial network by parametric and structural mutation. Beside this so called complexification, the key features of NEAT are the historical marking of new mutational innovations. With this mechanism crossover can be enabled by aligning the genomes and comparing the innovation history. Furthermore, to protect new structures, a niching mechanism, respectively speciation, is introduced. Again, the historical marking allows to calculate the distance between two different genomes and allows the classification of all members of the population with a configurable parameter into species.

The Evolutionary Acquisition of Neural Topologies (EANT) [12] also enables to evolve recurrent networks by parametric and structural mutation. The algorithm is based on the Common Genetic Encoding (CGE) [13] on which the mutation and a NEAT-like crossover operate. This genome encoding enables direct and indirect encoding, is complete, compact and closed. Additionally, they introduced the differentiation between an exploitation phase and an exploration phase. The exploitation phase only optimizes the existing structure by adapting the weights, without changing the structure of the network itself. The exploration phase allows the introduction of new genes by means of structural mutations. These two phases are alternating, starting with several exploitation steps followed by an exploration step so that networks with optimal structures, can adapt the weights of the links.

In Schlachter et al. [14] and Schwarzer et al. [15], we already demonstrated a neural network controller which incorporated structural evolution as well as the possibility to adapt incrementally to a dynamic environment. The advantages and the experiences are compared to all approaches and brought into the new framework.

III. Requirements

In order to evolve controllers for swarm and modular robots, some key issues, which should be fulfilled have to be addressed:

- **On-line and on-board evolution:** In addition to off-board and off-line evolution, the framework should be able to enable on-line and on-board evolution to met the requirements in the projects and allow a broad application scope.
- **Flexible Controller Types:** In order to enable the best choice for a certain scenario, the framework should support different types of control. Beside artificial neural networks it should allow to use spiking neural networks. Additional, other kinds of controllers should be easily integrable by a modular abstraction level.
- **Parametric and Structural Evolution:** Both the weights of links as well as the structure of a network need to be subject to mutational operators to allow complexification from a minimal initial network to the structure which is required by the task to fulfil and adjust the present weights.
- **Simple and flexible use:** The usability should be as simple as possible. The choices of controller type and parameters should be transparent and well organized.
- **Powerful interfaces:** The interfaces to required tools, simulations and the robot itself should be well suited to allow a complete use of the functionalities.
- **Application Range:** The evolutionary framework has to run on individual robots in a swarm, on reconfigurable robots in an artificial organism in a centralized as well as decentralized manner.

IV. Architecture and components

The framework is designed modular in object-oriented C++. It was carefully designed regarding reusability and extendibility. It supports dynamical changes of controllers during runtime, different network types, mutation operators, selection and fitness functions. An overview of the different modules can be seen in Figure 3. All modules are grouped into five associated groups, which are explained in more detail in the following.

A. Evolutionary Engine

The core of the framework is built by the evolutionary engine. This engine handles the population and the correct evaluation of it. It triggers the generation of the initial population and links the evolvable controllers with the modules for selection and fitness evaluation. Depending on the selection mode, it either processes generation by generation or allows for continuous tournament selection. Each population island consists of a configurable number of individual controllers, derived from the evolvable superclass.

B. EvoRoFConfig and Logger

The EvoRoFConfig module is responsible for the configuration files. In the initial phase, it reads in the files and sets

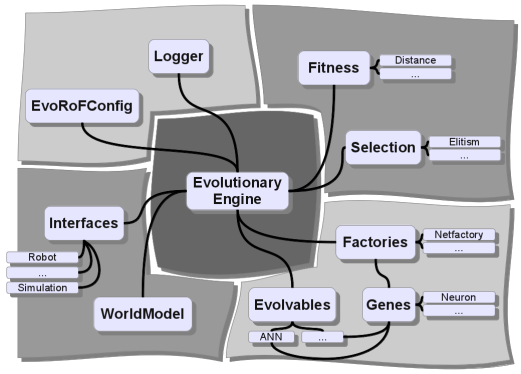


Fig. 3. Overview over the EvoRoF architecture. Central part is the evolutionary engine with the four surrounding blocks.

the configuration. During runtime, this module provides other modules with necessary information about the parameters required.

The logger gives a configurable interface to set the level of detail for logging. The complete framework is distributed in different log levels which can independently be switched on or off. In addition to program information and error states, this module takes care of logging of fitness values and genomes into files for later use or comparison.

C. Fitness and Selection

The modules for fitness and selection can be configured via the configuration file. The evolutionary engine coordinates their activities. The fitness module takes care of the correct evaluation of the fitness of the current running controller and stores the values. The selection mechanism, is responsible for the creation of the next generation, respectively of the next selected individual for evaluation. In generation based mode, this module generates depending on the selection scheme the next generation and delivers it back to the evolutionary engine. In tournament mode, the next individual will be generated and given back to the evolutionary engine.

D. Evolvables, Genes and Factories

Each controller is a subclass of the evolvable superclass. This class delivers the template to be implemented in order to be used by the evolutionary engine in the right way. All controllers have to implement the same interfaces like *initialize()* or *mutate()*.

Every controller encapsulates its own genome, which genes are derived from the genes class. Figure 4 shows an exemplary class hierarchy for the CGE genome.

To separate the creation of new controllers from the logic of a controller, several modules following the factory pattern are available. Those factories generate depending on the desired configuration the individual controllers and push them into the island population of the evolutionary engine.

E. Interface to Simulation and Real Robots

The EvoRoF framework should be able to address the relevant robots and simulation environments of the directly linked projects, thus be extendible to several platforms. For the ongoing experiments, we support interfaces, called wrappers, for the use with different simulators and the three available robot types in the Symbrion and Replicator projects. The used simulators are PlayerStage and the Robot3D simulator of the projects (see also VI Applications). The robot interfaces use the Symbricator Robot API [7]. For generic use, the evolutionary engine can be accessed with a plain wrapper.

Based on the interface functions a common worldmodel is implemented. This worldmodel serves as a container for all relevant sensor data and information from both internal and external sensors and states. In addition, the worldmodel takes care of the message processing of incoming and outgoing messages from and to other robots.

V. Implemented Concepts

A. Controller Types

A straight forward choice of the controller type for this kind of application is the use of neural networks. Following the definition of Maass [16] there are three classes of neural networks. The first generation is based on McCulloch-Pitts neurons (only digital output). These models can give digital output and are universal for every boolean function. The second generation is weighting the inputs and calculating the output via an activation function which delivers a continuous output value. The activation functions can vary from piecewise linear to sigmoid or even more complex functions. The network structure can be either a perceptron or a recurrent network. They can cope with analogue input and are universal for analogue computation. The third class of neural networks are the spiking neural networks. While in the second generation of networks, the output can be interpreted biologically as the current firing rate (number of spikes per period), the timing of spikes is in the foreground for spiking neural networks. The information can be encoded in the timing of spikes.

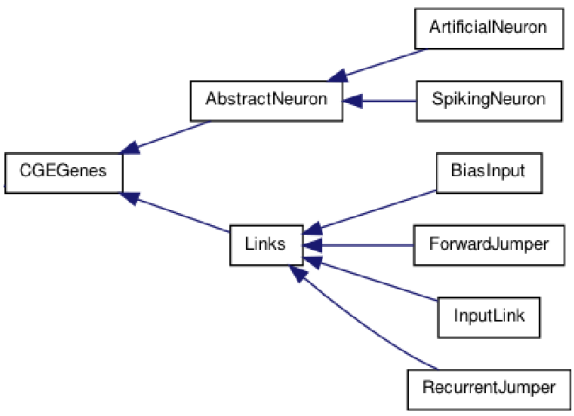


Fig. 4. The class hierarchy of the CGE gene classes.

In this framework, we implemented support for all types of networks in order to allow a higher flexibility in choice. Depending on the scenario and computational demands, the hidden layers can be disabled and even so recurrent connections can be switched off. Thus, the complete range from simple perceptrons to complex recurrent neural networks is feasible.

B. Genotype and Phenotype Representation

For the genotype we adopted the idea of the common genetic encoding (CGE) [13] which is also used in the EANT framework, described by Kassahun et al. [12]. The CGE is a linear genome representation and is in comparison to other approaches like GNARL or NEAT, complete, closed and modular. It further supports direct and indirect encoding and allows for direct evaluation of the genotype without decoding it to a phenotype. The structure of the network is implicitly given and due to its linear nature it is simple to serialize for transmission in order to exchange genomes.

The initial population can be selected as proposed in NEAT without hidden nodes and all inputs connected to all outputs or with an additional initial hidden layer. It is mandatory to start with a minimal configuration in order to find a minimal solution by continuous complexification. Alternatively, the EANT approach, starting with the same minimal network, but increasing the diversity of the start population by some random initial mutations, can be chosen.

C. Evolutionary Operators

The evolutionary operators allow to mutate both the weight parameters of links and the structural complexification by adding new links and nodes. Due to the nature of the CGE, either a forward jumper, a recurrent jumper or a complete subgenome with an arbitrary number of incident input connections can be inserted. The recombination is as described by Stanley et al. [11] in the NEAT framework. The genomes are aligned and combined to generate a new structure containing the common parts as well as the differing parts of both parents. Instead of the global tracking numbers for innovations, the identifiers can be used.

D. Evolutionary process, Fitness and Selection

To better support the on-line and on-board capabilities of the evolutionary framework, we adopted the idea of island evolution from [17]. Each robot represents an island with its own population. The population consists of configurable size of genomes. In the tournament selection mode, one or two genomes, depending if mating is enabled, are taken to generate new offspring. This new individual is then evaluated for a certain time and the fitness value is compared to the existing members of the island. If the fitness is higher as the worst member, this one will be replaced by the new genome. In the generation based mode, all genomes on an island are evaluated. Afterwards, the new offspring is generated. Depending on the configuration this could be for example elitism selection which allows only the children of the best 50 per cent to create the offspring for the next generation.

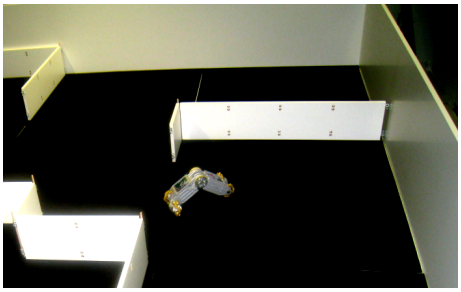


Fig. 5. An Active Wheel robot controlled by EvoRoF performing collision avoidance in a maze-like arena.

VI. Applications

The described framework is used in several application scenarios. It has been shown, that it is powerful enough, to be used in coevolution as well as in distributed on-line evolution for organisms control.

A. Evolution of Collision Avoidance

In this scenario, we used a prototype of the Active Wheel developed in the Symbrion and Replicator projects. We only used the IR sensors on the front and back side. The IR sensors on the back have to be taken into account, because the Active Wheel can collide with the back when turning due to the omnidirectional locomotion.

The population size was 15, the controller type a standard artificial network. There were six IR sensor inputs from the front and additional six sensors at the back extended by bias neuron. In all test runs, the Active Wheel evolved a collision free locomotion and walked randomly through the maze 5.

B. Coevolution of Coordinated Behavior

In a further experiment, we wanted to see the capabilities of the evolution of coordinated behaviour of robots [18]. For this reason, we set up a scenario with two target zones, in which both robots have to be at the same time to gain fitness. Beside a collision free locomotion, the robots have to develop a coordinated locomotion strategy in order to be in this target zones at the same time. Figure 6 shows the two robots, both the blue and red one, and the two yellow target zones in PlayerStage [19].

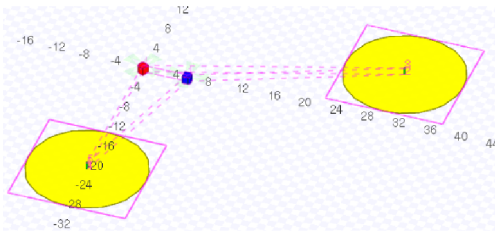


Fig. 6. The scenario: Two robots (blue and red) shall move in a coordinated manner from the left yellow power source to the one on the right upper side. Once a power source is “harvested” the robots have to move to the opposite target. Only when both robots are there, they gain power, respectively can increase their fitness.

 ISBN: 978-1-61208-274-5

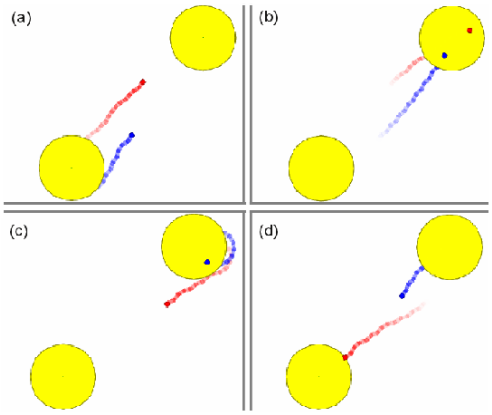


Fig. 7. The sequence shows the final behaviour of a run in alphabetical order in time. The red and blue robots start in the bottom left corner, reach the upper right corner and go back in a coordinated fashion.

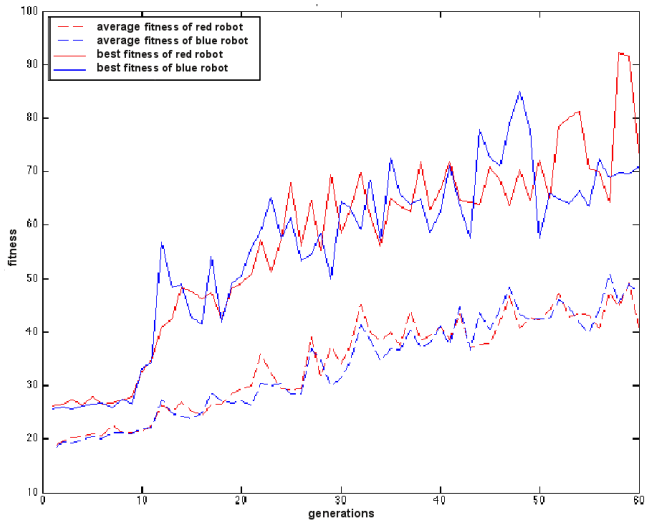


Fig. 8. Fitness development in the scenario. The graphs show the average fitness of both robots (doted lines) and the best individual per generation (solid line).

An evolved behaviour can be seen in Figure 7 where the position tracking of the two robots is depicted. In (a) they approach the upper target, in (b) they reached it and turn around in (c) to approach the lower yellow target in a coordinated way (d).

The used controller type was a recurrent neural network of class two using a cubical robot imitating the Backbone, respectively the Scout robot, of the Symbrion and Replicator projects. The input sensors were two front and two rear IR sensors and eight virtual sensors measuring the distance to the other robot and target zones. The population size was 10 in a generation-based run with different settings regarding the use of hidden neurons and the use of structural mutation. Figure 8 shows an average fitness development in a treatment with structural mutation enabled.

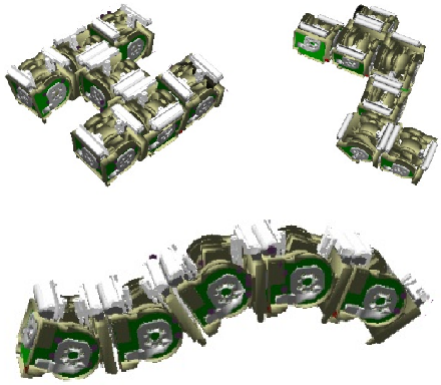


Fig. 9. Different types of multi-robot organisms tested in simulation.

C. Evolution of CPG control

In [20], locomotion for a multi-robot based on a spiking neural network was evolved. By distributed evolution, the organism should be able to emerge a global organism locomotion, by evolution of local control on each individual robot. The basic concept was a central pattern generator scheme, in which the parameters of a sine wave are modified in order to incorporate the sensor input and status messages from other modules. The individual robots have to learn considering the sensor input, which phase shift and amplitude to perform the necessary local behaviour.

In Figure 9, three exemplary organisms in the Robot3D simulator [21] are shown. Beside a caterpillar, we evolved central pattern generated behaviour for several different organism morphologies. The population size of each robot was ten and spiking neural networks are structurally mutated over 30 generations for 800 ticks evaluation time. Figure 11 shows the individual hinge positions of the five robots in the caterpillar-like configuration in one of the evaluation phases. The resulting behaviour, emerged by the individual hinge movement, is depicted in the sequence of Figure 10. A caterpillar is moving from the centre towards the left side in order to leave the sight of view.

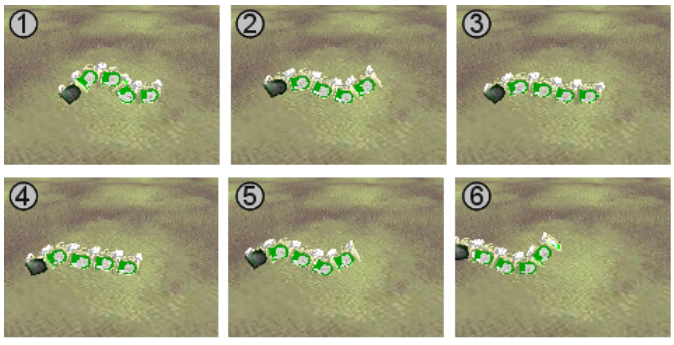


Fig. 10. The sequence shows the final behaviour of a caterpillar-like evolved locomotion.

 ISBN: 978-1-61208-274-5

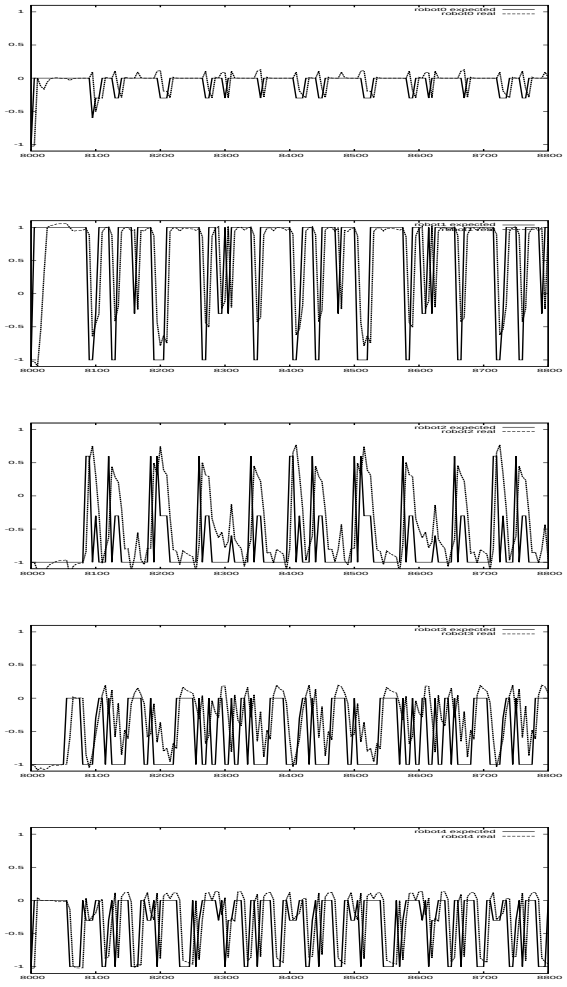


Fig. 11. The hinge positions in the evolved caterpillar-like robot organism.

VII. Conclusion

In this paper, we presented a framework for evolutionary robotics with the special focus on structural on-line and on-board evolution of neural network controllers. This framework supports standard neural networks as well as spiking neural networks in both generation based and tournament selection based evolutionary design processes. We have applied the framework to different scenarios in simulation and real robots to proof the feasibility. In future work, we will investigate the influence of structural mutation in more detail and will focus on the comparison of standard neural networks and spiking networks.

Acknowledgment

The "SYMBRION" project is funded by the European Commission within the work programme "Future and Emergent Technologies Proactive" under the grant agreement no. 216342. The "REPLICATOR" project is funded within the work programme "Cognitive Systems, Interaction, Robotics" under the grant agreement no. 216240.

References

[1] S. Murata and H. Kurokawa, "Self-reconfigurable robots," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 71–78, 2007.

[2] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 43–52, 2007.

[3] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology*. MIT Press, 2000.

[4] Symbrion: Symbiotic evolutionary robot organisms, 7th framework programme project no fp7-ict-2007.8.2, 2008-2013. http://www.symbrion.eu , visited on May 17th 2013.

[5] Replicator: Robotic evolutionary self-programming and self-assembling organisms, 7th framework programme project no fp7-ict-2007.2.1, 2008-2013. http://www.replicators.eu , visited on May 17th 2013.

[6] F. Schlachter, E. Meister, S. Kernbach, and P. Levi, "Evolve-ability of the robot platform in the symbrion project," in *SASOW: Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE Computer Society, 2008, pp. 144–149.

[7] F. Schlachter, C. Schwarzer, B. Girault, and P. Levi, "A Modular Software Framework for Heterogeneous Reconfigurable Robots ," in *P. Levi et al. (eds.), Autonomous Mobile Systems, AMS*, 2012.

[8] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, R. Humza, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi, "Multi-robot organisms: State of the art," *CoRR*, vol. abs/1108.5543, 2011.

[9] S. Kernbach, F. Schlachter, R. Humza, J. Liedke, S. Popesku, S. Russo, T. Ranzani, L. Manfredi, C. Stefanini, R. Matthias, C. Schwarzer, B. Girault, P. Alschbach, E. Meister, and O. Scholz, "Heterogeneity for increasing performance and reliability of self-reconfigurable multi-robot organisms," *CoRR*, vol. abs/1109.2288, 2011.

[10] P. J. Angeline, G. M. Saunders, and J. P. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, January 1994.

[11] K. O. Stanley and R. Miikkulainen, "Evolving neural network through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[12] Y. Kassahun, J. Metzen, M. Edgington, and F. Kirchner, "Incremental acquisition of neural structures through evolution," in *Design and Control of Intelligent Robotic Systems*, ser. Studies in Computational Intelligence. Springer, 2009, pp. 187–208.

[13] Y. Kassahun, M. Edgington, J. H. Metzen, G. Sommer, and F. Kirchner, "A common genetic encoding for both direct and indirect encodings of networks," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1029–1036.

[14] F. Schlachter, C. Schwarzer, S. Kernbach, N. K. Michiels, and P. Levi, "Incremental online evolution and adaptation of neural networks for robot control in dynamic environments," in *ADAPTIVE: Conference on Adaptive and Self-Adaptive Systems and Applications*, 2010, pp. 111–116.

[15] C. Schwarzer, F. Schlachter, and N. K. Michiels, "Online evolution in dynamic environments using neural networks in autonomous robots," *Internation Journal on Advances in Intelligent Systems*, vol. 4, no. 3&4, 2011.

[16] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, pp. 1659–1671, 1996.

[17] N. Bredeche, E. Haasdijk, and A. Eiben, "On-line, on-board evolution of robot controllers," in *Proceedings of the 9th international conference on Artificial Evolution (Evolution Artificielle - EA'09)*, 2009.

[18] K. Deuschl, "Evolution of coordinated behavior in a heterogenous robot swarm," *Diploma Thesis*, University of Stuttgart, 2012.

[19] "Playerstage," Website, 2013, available online at https://launchpad.net/robot3d; visited on May 17th 2013.

[20] P. Alschbach, "Online evolution and adaptation of central pattern generators for multi-robot organisms," *Diploma Thesis*, University of Stuttgart, 2012.

[21] "Robot3d, open source modular swarm robot simulation engine," Website, 2013, available online at https://launchpad.net/robot3d; visited on May 17th 2013.

 ISBN: 978-1-61208-274-5

An Experimental Framework for Exploiting Vision in Swarm Robotics

Sjriek Alers, Bijan Ranjbar-Sahraei, Stefan May, Karl Tuyls and Gerhard Weiss
Department of Knowledge Engineering
Maastricht University
Email: {sjriek.alers,b.ranjbarsahraei}@maastrichtuniversity.nl,
stefan.may@student.maastrichtuniversity.nl, {k.tuyls, gerhard.weiss}@maastrichtuniversity.nl

***Abstract*—This paper studies the requirements of a successful vision-based approach in swarm robotic settings. Required features such as landmarks and different patterns are introduced, and appropriate feature detection algorithms are described in detail. The features are designed to be very simple, and providing enough information, while the proposed detection algorithms have considered the very limited resources (i.e., limited storage memory, and limited computational power) of swarm robots. In order to evaluate the performance of the proposed vision approaches and the defined features for the environment, the whole approach is verified by implementation on e-puck robots in a real-world setting.**

***Keywords*—Robot vision systems; Multirobot systems.**

I. INTRODUCTION

Natural phenomena have always fascinated and inspired scientists, not only the biologists but also others such as computer scientists. One of the interesting phenomena in nature is the behavior seen in colonies of social insects such as ants and bees. These insects have evolved over a long period of time and display a behavior that is highly suitable for addressing the complex tasks that they face. Therefore, over the recent years an increasing interest is seen among researchers for creating artificial systems that mimic such behavior for accomplishing the complex tasks that humans face in their life [1], [2], [3].

The phenomenon that intelligent behavior emerges from a collection of simple interactions among agents which are relative simple as well, is generally referred with the term Swarm Intelligence (SI) [4]. The best known example for emergence of Swarm Intelligence among social insects is the foraging behavior of ants. In ant foraging, ants deposit pheromones on their path during traveling. Using this path they are able to navigate between the nest and food [5]. A slightly different foraging behavior can be seen among honeybees. Instead of using pheromones to navigate through an unknown environment, honeybees use a strategy called *Path Integration*, in combination with landmark navigation [6]. With the aim to transfer such social behaviors to embodied systems, many researchers are investigating the foraging behavior of ants and bees, by using robots in real environments. However, Foraging is the task of locating and acquiring resources in an unknown environment, which is quite a difficult task, in terms of localization and detection of environmental localization, specially for simple robots in a distributed swarm. The foraging task can be seen as an abstract representation for many other advanced tasks, such as patrolling and routing. Therefore, a successful embodied implementation of distributed foraging can result in promising applications in, e.g., security patrolling, monitoring of environments, exploration of hazardous environments, search and rescue, and crisis management situations.

Getting motivation from the mentioned potential applications of distributed coordination and following the previous work [7], [8], in which we mainly relied on random exploration methods and infrared sensor data for obstacle detection, this paper is focusing on using vision for detecting key environmental features. These features then can be used as waypoints to navigate in an unknown environment, locate other entities, and detect modifications made in the environment.

For this purpose, we explore several visual features that can be used for acquiring information from the environment by a robot with limited computation abilities, and equipped with a simple camera. For detecting key locations in the environment (e.g., corners in a maze), we investigate the usage of specific landmarks for these locations. Each landmark consists of an upper ring with a solid color, so that it can be detected from a distance, and on the lower part a unique barcode for keeping track of the landmark numbers. Furthermore we explore the possibility to detect markers with an even higher data density: QR-codes. The challenge in the detection of these two-dimensional codes, lies in analyzing and processing the camera data with the limited processing and memory resources that are available in our robotic platform. Finally, the most common feature already available in every robotic swarm setting is the robot itself. It's always favorable to detect the relative location and orientation of other robots in respect of one's position. Therefore, the available LEDs on the robot provide a very good feature for robot detection from a distance. Moreover, we have designed specific gradient patterns for nearby robot detection, which can conclude to a very accurate orientation detection.

Authors believe that the proposed environmental features defined in this paper, in combination with the detection algorithms which are included as well, can provide an experimental framework for any kind of swarm robotic experiment with simple robots (e.g., [9], [7], [8], [10], [11]) as illustrated in Fig. 1.

The remainder of the paper is structured as follows: The physical setup and designed software are described in Section II. The main features used in this paper are defined in Section III, and the techniques for detection of each feature is described in Section IV. A real-world demonstration of this work is

 ISBN: 978-1-61208-274-5

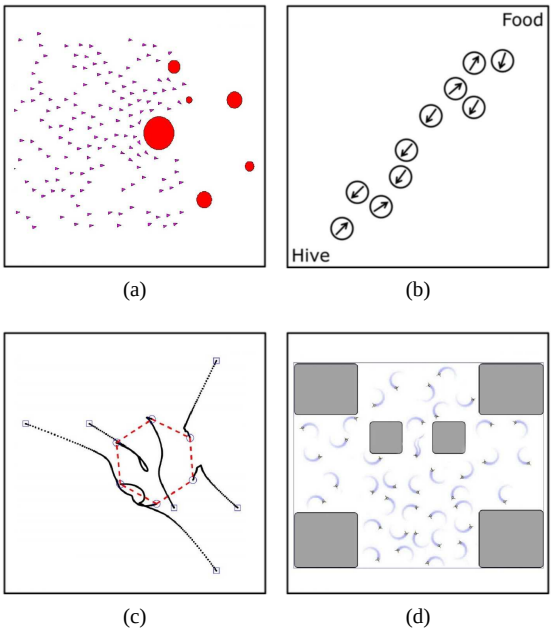


Figure 1. Different swarm robotic applications which require visual feature detection: (a) Flocking in multi-agent systems [9]. (b) Bee-inspire foraging [7]. (c) Formation control in multi-agent systems [10]. (d) StiCo: Stigmergic coverage in complex environments [11].

described in Section V and can also be found in [12]. Finally, in Section VI we will give the concluding remarks and future work.

II. Physical Setup and Designed Software

The e-puck robot is a small platform for educational and research purposes, developed by the EPFL University [13]. This robot is efficiently used in numerous projects in the domain of swarm robotics and swarm intelligence (e.g., [7], [8], [14], [13]).

The main features of the e-puck robot include, but are not limited to; a robust design, flexibility for a large spectrum of educational activities, compact size, and rich on-board accessibilities (e.g., microphones, accelerometer, camera).

In this section, first the hardware specifications of the e-puck are briefly introduced, then the developed software which is designed for monitoring the e-puck camera during its image processing and feature detection tasks is described.

A. Hardware Specifications

The e-puck hardware consists of different sensor types for detecting visible or Infra Red (IR) light, sound, acceleration, et cetera. The motors are the only actuators which are available in e-puck. A microprocessor of PIC family with 8 KB RAM memory, assist the robot to get data from it's sensors, analyze it, and perform actions. The main hardware elements, which are involved in our experiments are listed in Table I.

As listed in the table, the on-board camera of the e-puck has a resolution of 640×480 pixels. It is placed at the front of the e-puck, 2.7 cm above the floor. With this camera, objects that are placed on the floor can be detected at a minimum distance of 7.4 cm. The camera angle is approximately $40°$,

TABLE I. E-puck technical specification

Element	Technical information
Processor	dsPIC30F6014A @ 60 MHz (15 MIPS), 16-bit microcontroller with DSP core
Memory	RAM: 8KB Flash: 144 KB
Motors	2 stepper motors with a 50:1 reduction gear
Camera	VGA color camera with resolution of 640x480 pixels
LEDs	8 red LEDs on the ring, green LEDs on the body, 1 high intensity red LED in the front
Wireless Communication	Bluetooth for robot-computer and robot-robot communications Infrared for robot-robot communication

and at this minimum distance, objects of 5.1 cm width can be fully seen.

Remark 1: *Although, we have a VGA camera, the on-board processing and storage of the e-puck robot is not adequate for dealing with all of the camera data. A gray-scale image of size* 640×480 *needs at least* 307.2 *KB to store the image. However, based on the technical details of Table I, the e-puck robot has a RAM of size* 8 *KB. Analysis, and storage of sub-parts of the image helps to overcome this limitation. In following sections of this paper, we address the issue of how to split an image into informative sub-parts.*

B. Software

In order to monitor the e-puck in real-time, and for debugging the image-processing algorithms, a Java-based software application is developed. This software, shown in Fig. 2, communicates with the e-puck via Bluetooth. It receives text messages from the e-puck, which are reports of intermediate statuses of the e-puck (e.g., "found something", "driving to the landmark", "code read", "searching"). At the same time, the program also receives the captured image from the robot. Logging all of the data, storing the text messages and captured images, as well as the filtered and segmented images, makes both real-time and offline debugging very easy. Finally, it should be mentioned that time stamps are always attached both to the captured images, and stored text messages.

Figure 2. Developed software for monitoring the e-puck: (1) The required controls to establish the connection with the e-puck. (2) The real-time captured image. (3) Log statements (4) Archive of the last 20 captured pictures.

 ISBN: 978-1-61208-274-5

III. FEATURE DEFINITION

Defining a collection of detectable features is, due to the limited resources of a simple robot, an important task that is a part of the main scope in this paper. Different objects in the environment (e.g., pieces of wood, balls, walls, floor, and robots) and many available patterns (e.g., different colors, checkerboard and barcodes) can be considered as environmental features, however their detection via a robot with limited capabilities might be computationally complex, and or not adequately robust to environmental disturbances (e.g., light variations and distance variations).

Generally, we define the required environmental features into two main categories: *far features*, and *close features*. For the far features bright lights (e.g., from a red LED) or specific relative large areas with solid colors, should be considered. Moreover, these features should be recognizable from different directions, which makes cylindrical shapes more favorable. However, for nearby features, the patterns which can store higher amount of information are required (e.g., a one-dimensional barcode or two-dimensional codes which can store digital information). By considering the mentioned constraints, a collection of the most appropriate environmental features will be introduced in this section.

A. Landmark

Most important features for an environment are the landmarks. Robots can use landmarks in many various missions, like localization, mapping, exploration, etc cetera. These features should be recognizable from different directions, and also from a distance. Landmarks should provide useful information to the robots (e.g., their exact location), therefore, we introduce a cylindrical tube, as shown in Fig. 3, which is a combination of a colored ring and a barcode.

At the top of the cylinder a colored ring is denoted which is easily detectable from a distance. For our setup, purple is chosen as ring color, as purple is a color that does not exist in any other objects in our environment. To differentiate the landmarks, an EAN-8 (European Article Number) barcode was selected, containing an ID consisting of 8 digits, including a control number. The EAN-8 barcode is printed vertically below the purple block, surrounding the whole cylinder.

B. QR-Code

Although, landmarks are very useful in terms of being detectable from a distance, we need a smaller pattern which can be mounted on walls, and also directly on robots for providing more dense information (e.g., specific ID of a robot, wall orders). Therefore, we use a two-dimensional Quick Response code (QR) which is developed as a universal data storage standard. These QR-codes can store a higher data density, then the EAN-8 barcode.

The only disadvantage of the QR-code is the complexity of its pattern. In general, the pattern is comprised of several parts: At the top left, the top right and the lower left corner an orientation pattern is placed. It is a square of size 9×9 modules. The fourth corner does not contain this pattern, which makes detection of QR-code angle easier. In most swarm robotic applications, the orientation of the QR-code can be fixed, so the orientation check can be ignored during image-processing, decreasing the computational complexity drastically.

Different versions of QR-codes have different sizes. The smallest size is Version 1, which has a size of 21×21 modules. For each version, the size is increased by 4 modules in each direction. Between the three orientation patterns there are timing pattern lines with strict changing modules of black and white at row 6 and column 6. Every QR-code from Version 2 and higher, contain position adjustment patterns at specific points. In Fig. 4 the structure for QR-code Version 3 is given, in which the black and white parts are fixed.

C. Robot Detection

A very important feature which will be available in the environment of any swarm robotic application, is the robot itself. Inherently, the robots contain various information, like their position, orientation, and their identifier, which can be very useful for the other individuals to know. Therefore, the ability to detect other robots relative orientation and location, is very convenient for implementation of many complex swarm algorithms (e.g., [7], [8]).

In practice, a good way for detecting other robots with a camera, is detecting the robots light sources (e.g., on-board LEDs). As such a light source has a good contrast to the other parts of the environment, it can be detected from a far distance even on low resolution captured images.

To determine the orientation of the robots, based on it's visual features (e.g., the wheels and body of the robots) is a really complex task. Therefore, we propose to add a black-white pattern comprised of two slopes as shown in Fig. 5. Computing the exact orientation of a robot by using this pattern is easily implementable.

Based on the standard size of an e-puck robot, the pattern should have a total height of 33 mm and consists of two black bars separated by a white bar on top. All the bars are 3 mm

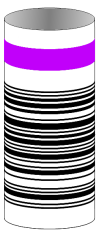


Figure 3. Example for a landmark

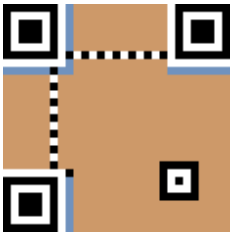


Figure 4. Structure of a QR-code Version 3, displaying orientation and timing patterns

Figure 5. Body pattern of e-puck for orientation detection

in height. A sloped pattern, in the form of a black triangle is added to the bottom of the pattern.

Finally it should be mentioned that, in addition to orientation measurement, distance measurement also becomes possible by using this specific pattern.

IV. Feature Detection

In the previous section, we introduced four main features: *landmark*, *QR-code*, *Robot LED*, and *Robot body pattern*. The main approach for detection of these features is to first use basic filters for highlighting the required information (e.g., purple color or edges in the image) and then zooming into the informative part of the image for reading it in more details. The most important factor in designing each detection algorithm, is to use the least possible memory and computation power. In the following subsections, these techniques for detection of each feature will be described.

Remark 2: *It should be mentioned that all of the required thresholds which will be used in following subsections are computed based on practical experiments and with real-time calibrations. However, describing these experiments in detail is beyond the scope of this paper.*

A. Landmark

The landmark contains a purple ring and an EAN-8 code. The landmarks are designed to be taller than robots. Therefore, finding the purple ring of each landmark, limits the scanning area of the image to the upper half of the camera view.

Detection of an area with a specific color is a basic task. In the first step a color filter with the specific color is applied on the image. Resulting in a grayscale image with bright values for the colors which match the color the best. To avoid errors where single pixels fit to the color, the image is blurred with a Gaussian algorithm [15]. Afterward the image is split into a binary black/white image with a fixed threshold. This procedure is illustrated in Fig. 6.

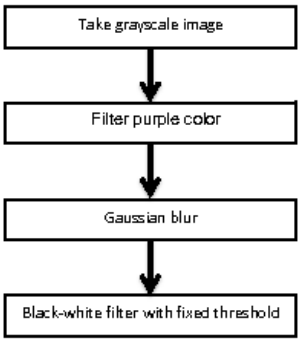


Figure 6. The required pre-processing procedure for detection of a specific color (e.g., purple).

After this pre-processing phase, a group-finding algorithm [15] is applied on the image, and the largest group, is considered as the purple ring.

Consequently, the exact position of purple ring in the image can help to estimate its distance to the robot. The higher the purple ring is, the further the distance should be. This estimated distance is used to find the appropriate distance to start reading the barcode. As soon as the required distance, in which the barcode is readable is reached, the required scanning area is determined (i.e., the area under the purple ring).

Barcodes are one-dimensional, this simplifies the scanning process and makes the whole procedure faster. Therefore, addressing the issue described in Remark 1, the robot prepares a grayscale image with low resolution in width but high resolution in height (i.e., zooming into an area of 4 pixels in width and 80 pixels in height).

The pre-processing for the EAN-8 barcode is done by using a halftone filter [15] with a threshold calculated by an average of the pixels intensity. Afterward, all patterns of form black-white-black, as shown in Fig. 7 are located. Based on the EAN-8 standards, at least three occurrences should be detected for the start, center and end of barcode.

After this validation check, the part of the image containing the code is transformed into the 67 bits representing the barcode. Each bit is defined by the average of the pixels it represented. For each seven bits the best corresponding match to a data character is determined. As a last step, the control-character is calculated out of the seven data-characters.

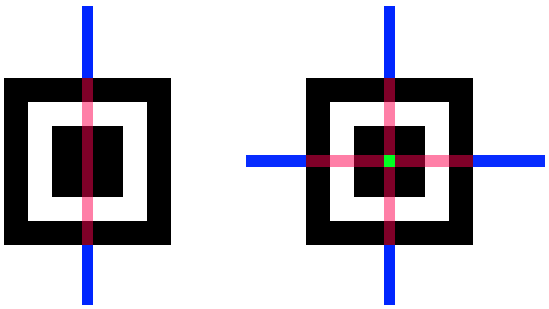
B. QR-Code

Detection of QR-codes is more complex than detection of one-dimensional barcodes. We assume that the QR-code is fully visible in the camera frame, as a partial QR-code cannot be decoded. As the QR-code needs a resolution as high as possible, first a black-white image is filtered out of the initial captured image. For finding the three orientation markers of the QR-code, a pattern finding algorithm is used, which looks for a black-white-$3\times$black-white-black transition on each column. The detectable pattern looks like the center line in Fig. 8a. As soon as the pattern is found, the same pattern is located in the rows. The results should be similar to Fig. 8b. The orientation marker validation is passed, if both found regions have nearly the same size and center.

A QR-code is comprised of a collection of modules, each black or white. In order to determine the number of pixels which construct a single module, the size of the orientation modules, and their distance to each other can be used. For example, the pattern shown in Fig. 8 consists of 7 modules, so we can divide the number of pixels in this pattern by 7 to compute the size of a single module. Moreover, to improve

Figure 7. Structure of EAN-8 barcode, with the black-white-black pattern in the beginning, middle, and end.

 ISBN: 978-1-61208-274-5



(a) (b)

Figure 8. Detection of QR-code orientation markers: (a) pattern detected in vertical alignment. (b) pattern detected in horizontal alignment.

the estimation of the module size, the distance between two patterns can also be used. Each version has a size of $21 + n \cdot 4$ modules, where n is the version of QR-code. Therefore we can calculate the version and get a more exact value for the module size. The decoding process of the QR-code after its structure has been extracted is described in [16].

The most challenging problem with an QR-code image, is that the image has to be stored in a high enough resolution, for being decodable. However, addressing the issue mentioned in Remark 1, the memory on the e-puck is limited for storing additional information to 4 KB for which each bit can store one pixel, as the rest of the memory is used for running the algorithms. To have some error tolerance, there should be at least four pixels describing one module. Therefore, we can find the biggest detectable size for QR-code with following equation:

$$4000 \times 8 = modules^2 + (4 \cdot modules)^2 \qquad (1)$$

in which the left side shows number of available bits, and on the right side, the first and second terms show the number of required bits for storing the QR-code and image itself. This equation concludes to the fact that width and height of the QR-code should not exceed 43×43 modules. The QR-code version which fits into 43×43 modules is Version 6, which is 41×41. In practice we also need memory for the detection algorithms and internal calculations, so the QR-code Version 5 which contains 37×37 modules, is used in our experiments.

C. Robot Detection

An other robot is generally detected in two different steps. First, the detection from a distance is done by searching for the red LEDS, and second, when nearby, the body pattern (Fig. 5) which consists of two black ramps around the robot, is scanned for measuring the exact orientation of robot.

1) LED Detection: The LEDs are mounted above the camera on the e-puck. Therefore only the upper half of the image has to be scanned, which results in a higher usable resolution of the relevant parts of 20×80 gray-scale pixels. First, a black-white filter with a fixed threshold is used. The threshold is chosen to be higher that ambient light, and less that the brightness of an LED. The next filter is a Gaussian blur filter, which is used to combine light groups that are very close to each other, and dismisses single pixels that are falsely

(a) (b) (c) (d) (e) (f)

Figure 9. Different steps of pre-processing for LED detection.

recognized. Afterwards, a black-white filter is used, this time with an average threshold. All LEDs are now highlighted.

As it is not possible with this simple detection technique to really determine between a red LED, and LEDs from another color, some improvements should be applied. Therefore, after detecting the light sources, the camera zooms in on each group center (zooming is a built-in feature of the e-puck camera). The zooming ratio depends on the amount of pixels that belong to each group. In Fig. 9 an image of a zoomed in LED is shown. There is a bright center visible with red at the left and the right, but not at top or bottom. This is a result of the surrounding border of the e-puck. To verify that the LED is a red one, both sides of the detected light source, starting from the center are scanned for a red color. The color is checked, by converting the image into the HSL color space and comparing the Hue value, as the lightness and saturation are very unstable. We consider a light source as an LED, if more than 50% of the height of the bright center contains a red surrounding.

2) Body Pattern Detection: If the robot is located close enough to the camera, the body pattern detection can be activated. To get the highest probability to detect the e-puck, the image should have a high resolution, but still work fast. The maximum image size which fits into memory and leaves enough space for the other required operations, is 80×40 pixels in gray-scale. As pre-processing step, a black-white filter with average threshold is applied on the image.

Subsequently, for each column of the image a pattern with one white, and one black module is located. For all locations where the pattern fits, a check is performed if the repetitive white and black modules have approximately a size of 5. If this holds, the column is stored as a part of the pattern. Fig. 10a shows a captured image from an epuck, and Fig. 10b highlights the parts of image which are extracted as body pattern according to this technique.

For rejecting wrong detections, only modules with at least three detected neighboring results are accepted. Afterward, the center of the e-puck is determined by searching the location where most left and right results are found and dividing their x-coordinates by 2. The orientation of robot can be easily measured by computing the length of middle white module, and comparing this size, with the size of the whole pattern.

(a) (b)

Figure 10. Body patter detection (a) Initial image. (b) the detected pattern is highlighted.

 ISBN: 978-1-61208-274-5

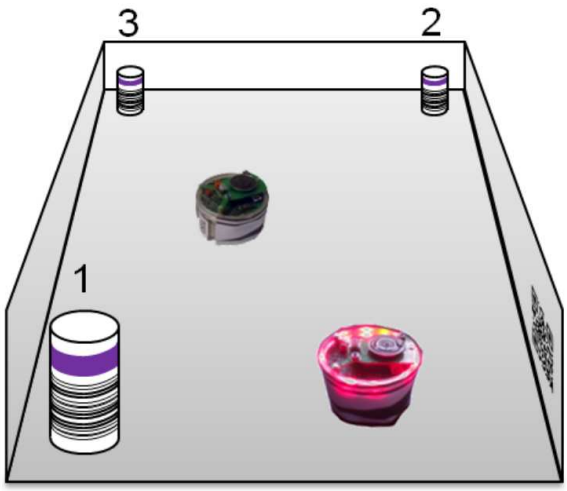


Figure 11. Designed scenario for validation of proposed approach

V. Real World Demonstration

To examine the proposed approach in a real scenario, an environment as shown in Fig. 11 is set up: A white floor of 40×40 cm^2 is surrounded with white walls. Three landmarks are placed in three corners, and in the fourth corner a QR-Code is attached to the wall. Two e-pucks are placed in this environment. One is stationary, with all of the red LEDs on, and a body pattern around it. The second robot uses the vision-based detection algorithms for detecting the features of the environment.

In this scenario, the robot has to first located landmark #1, continue to #2 and then drive to #3 in the correct order. For each landmark it has to approach it, read the barcode, and after validating the number find the other robot. By using the other robots orientation, it should move in the environment till both robots are facing each other from the front. Finally, the QR-code mounted on the wall is detected, and the code will be extracted.

A video of this performed experiment can be found in [12], including the preprocessed image data sent from the robot.

VI. Conclusions and Future Work

In this paper, we proposed a feature detection approach based on robot vision, which can be useful for swarm robotic experiments. The e-puck robot was chosen as the main platform for doing experiments. This robot is equipped with a VGA camera, but has limited resources for storing data, and also in performing computations. Therefore, different possible environmental features were introduced, and described accurately. Afterward, required image processing techniques for detection of each feature were described in detail. Finally, a general demonstration was set up to show the applicability of the proposed approach in a real-world robotic experiments.

The feature detection approaches developed in this paper, provided a framework for exploiting vision in various multi-robot scenarios. However, the performance of this framework was not sufficiently evaluated. Therefore, as a future work, authors will examine the introduced techniques accurately, by applying them in different experimental conditions (e.g., light or observation distance variations). Besides, the framework will be applied to a real swarm of robots (e.g., 7 e-puck robots). Finally, extending the framework by introducing other environmental features such as glowing artificial pheromones is a part of this project.

References

[1] F. Dressler and O. B. Akan, "A survey on bio-inspired networking," *Computer Networks*, vol. 54, no. 6, pp. 881 – 900, 2010.

[2] D. Floreano and C. Mattiussi, *Bio-inspired artificial intelligence: theories, methods, and technologies*. The MIT Press, 2008.

[3] N. Franceschini, F. Ruffier, and J. Serres, "A bio-inspired flying robot sheds light on insect piloting abilities," *Current Biology*, vol. 17, no. 4, pp. 329–335, 2007.

[4] J. Kennedy, "Swarm intelligence," *Handbook of nature-inspired and innovative computing*, pp. 187–219, 2006.

[5] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851–871, 2000.

[6] N. P.-P. M. Lemmens, *Bee-inspired Distributed Optimization*. Maastricht University, 2011.

[7] S. Alers, D. Bloembergen, D. Hennes, S. de Jong, M. Kaisers, N. Lemmens, K. Tuyls, and G. Weiss, "Bee-inspired foraging in an embodied swarm (demonstration)," in *Proceedings of the Tenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2011)*, pp. 1311–1312, 2011.

[8] N. Lemmens, S. Alers, and K. Tuyls, "Bee-inspired foraging in a real-life autonomous robot collective," in *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pp. 459–460, 2011.

[9] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[10] B. Ranjbar-Sahraei, F. Shabaninia, A. Nemati, and S. Stan, "A novel robust decentralized adaptive fuzzy control for swarm formation of multiagent systems," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 8, pp. 3124–3134, 2012.

[11] B. Ranjbar-Sahraei, G. Weiss, and A. Nakisaee, "A multi-robot coverage approach based on stigmergic communication," in *Multiagent System Technologies*, vol. 7598 of *Lecture Notes in Computer Science*, pp. 126–138, Springer, 2012.

[12] Swarmlab, Maastricht University, "Demonstration of an experimental framework for exploiting vision in swarm robotics." http://swarmlab.unimaas.nl/papers/adaptive-2013-demo/.

[13] F. Mondada, M. Bonani, *et al.*, "The e-puck, a robot designed for education in engineering," in *9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, pp. 59–65, IPCB: Instituto Politcnico de Castelo Branco, 2009.

[14] A. Breitenmoser, M. Schwager, J. Metzger, R. Siegwart, and D. Rus, "Voronoi coverage of non-convex environments with a group of networked robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4982–4989, 2010.

[15] R. Gonzalez and R. Woods, *Digital image processing*. Prentice Hall Upper Saddle River, NJ, 2002.

[16] "ISO/IEC 18004:2000, information technology, automatic identification and data capture techniques, bar code symbology, QR code," 2000.

 ISBN: 978-1-61208-274-5