ADAPTIVE 2014

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

ISBN: 978-1-61208-341-4

May 25 - 29, 2014

Venice, Italy

ADAPTIVE 2014 Editors

David Musliner, SIFT, LLC, USA

Elena Troubitsyna, Abo Akademi University, Finland

Dan Tamir, Texas State University, USA

ADAPTIVE 2014

Foreword

The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2014), held between May 25-29, 2014 in Venice, Italy, 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 2014 Technical Program Committee, as well as all of the reviewers. The creation of such a 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 2014. 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 2014 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that ADAPTIVE 2014 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the area 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 Venice, Italy, provided a pleasant environment during the conference and everyone saved some time to enjoy the charm of the city.

ADAPTIVE 2014 Chairs:

Radu Calinescu, University of York, 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
Marc Kurz, Johannes Kepler University Linz - Institute for Pervasive Computing, Austria
Dalimír Orfánus, ABB Corporate Research Center, Norway
Weirong Jiang, Xilinx Research Labs, San Jose, USA
Kier Dugan, University of Southampton, UK

ADAPTIVE 2014

Committee

ADAPTIVE Advisory Chairs

Radu Calinescu, University of York, 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
Marc Kurz, Johannes Kepler University Linz - Institute for Pervasive Computing, Austria

ADAPTIVE Industry/Research Chairs

Dalimír Orfánus, ABB Corporate Research Center, Norway
Weirong Jiang, Xilinx Research Labs, San Jose, USA

ADAPTIVE Publicity Chairs

Kier Dugan, University of Southampton, UK

ADAPTIVE 2014 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 M. Alcaraz Calero, University of the West of the Scotland, 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
Bernhard Bauer, University of Augsburg, Germany
Imen Ben Lahmar, Institut Telecom SudParis, France
Christophe Bobda, University of Arkansas, USA
Jean Botev, University of Luxembourg, Luxembourg
Jesus G. Boticario, Spanish National University for Distance Education (UNED), Spain
Sven Brueckner, Axon, USA
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

Henrique Lopes Cardoso, LIACC, Universidade do Porto, Portugal
Emiliano Lorini, Institut de Recherche en Informatique de Toulouse (IRIT), France
Sam Malek, George Mason University, USA
Paulo Martins, University of Trás-os-Montes e Alto Douro (UTAD), Portugal
Olga Melekhova, Université Pierre et Marie Curie - Paris 6, France
Frederic Migeon, IRIT/Toulouse University, France
Gero Müehl, University of Rostock, Germany
Christian Müller-Schloer, Leibniz University of Hanover, Germany
Masayuki Murata, Osaka University, Japan
Filippo Neri, University of Naples "Federico II", Italy
Dirk Niebuhr, Clausthal University of Technology, Germany
Andrea Omicini, Università di Bologna, Italy
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Mathias Pacher, Leibniz Universität Hannover, Germany
Thanasis Papaioannou, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
Alexandros Paramythis, Contexity AG, Switzerland
Georg Püschel, Technische Universität Dresden, 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
Estefanía Serral, Vienna University of Technology, Austria
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
Christof Teuscher, Portland State University, USA
Peppo Valetto, Drexel University, USA
Arlette van Wissen, VU University Amsterdam, Netherlands
Eiko Yoneki, University of Cambridge, UK

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.

Table of Contents

Model-based Run-Time Software Adaptation for Distributed Hierarchical Service Coordination

Hassan Gomaa, Koji Hashimoto

Department of Computer Science
George Mason University
Fairfax, VA, USA
hgomaa@gmu.edu, kojihashi@gmail.com

Abstract **- Dynamic software adaptation addresses software systems that need to change their behavior at run-time. A software adaptation pattern models how the components that make up an architecture pattern cooperate to change the software configuration at run-time. This paper describes a model-based run-time adaptation pattern for distributed hierarchical service coordination in service-oriented applications, in which multiple service coordinators are organized in a distributed hierarchical configuration.**

Keywords: service-oriented architecture; dynamic software adaptation; model-based software adaptation pattern; hierarchical service coordination adaptation.

I. INTRODUCTION

Dynamic software adaptation addresses software systems that need to change their behavior at run-time [1]. With model-based dynamic software adaptation, models are used to describe and sequence the adaptation of the software architecture and executable system at run-time [2]. A model-based software adaptation pattern defines how the components that make up an architecture or design pattern dynamically cooperate to change the software configuration to a new configuration given a set of adaptation commands. Because control and sequencing is so important in dynamic run-time adaptation, this research focuses on dynamic models, using in particular state machine models and object communication models.

Previous work has described model-based adaptation patterns for distributed component-based systems [2] and service-oriented architectures (SOA) [3][4]. In typical SOA applications, services are self-contained, loosely coupled, and orchestrated by coordination services [8]. This research addresses dynamic adaptation based on SOA coordination patterns. Previous work addressed independent SOA service coordination [3] and transaction-based distributed software adaptation [4], in which there is one service coordinator orchestrating multiple services. This paper extends this research to SOA applications with hierarchical service coordination by describing and validating a dynamic software adaptation pattern for distributed hierarchical service coordination in which a higher-level coordinator communicates with multiple lower-level coordinators.

This paper describes related work in Section II, provides an overview of software adaptation for SOA in Section III, describes in detail the hierarchical service coordination adaptation pattern in Section IV, describes its validation in Section V, and provides concluding remarks in Section VI.

II. RELATED WORK

Dynamic software architectures and dynamic reconfiguration approaches have been applied to dynamically adapt software systems. Research into self-adaptive, self-managed or self-healing systems includes approaches for monitoring the environment and adapting a system's behavior in order to support run-time adaptation [11]. Kramer and Magee [1] describe how a component must transition to a quiescent state before it can be removed or replaced in a dynamic software configuration. Ramirez and Cheng [5] describe applying adaptation design patterns to the design of an adaptive web server. The patterns include structural design patterns and reconfiguration patterns for removing and replacing components.

For service-oriented computing and service-oriented architectures, Li et al. [9] describe an adaptable service connector model, so that services can be dynamically composed. Irmert et al. [10] provide a framework to adapt services at run-time without affecting application execution and service availability. A related research area is dynamic adaptation of software product lines, in which the different software configurations are organized as a product line, with dynamic adaptation from one member configuration to another managed through a feature model [6].

In comparison with the previous approaches, this paper focuses on dynamic self-adaptation in service-oriented architectures. This paper describes a software adaptation pattern for distributed hierarchical service coordination, in order to adapt not only services but also distributed hierarchical coordinator components.

III. SOFTWARE ADAPTATION FOR SOA

In SOA applications, services are intended to be self-contained and loosely coupled, so that dependencies between services are kept to a minimum. Instead of one service depending on another, it is desirable to provide coordination services (also referred to as coordinators) in situations where access to multiple services needs to be coordinated and/or sequenced [3].

 ISBN: 978-1-61208-341-4

A. Software Coordination and Adaptation

In SOA systems, loose coupling is ensured by separating the concerns of individual services from those of the coordinators, which sequence the access to the services. As there are many different types of service coordination, it is helpful to develop service coordination patterns to capture the different kinds of service coordination. For each of these coordination patterns, there is a corresponding dynamic adaptation pattern [3]. The software adaptation patterns described in this paper were developed as part of Self-Architecting Software Systems (SASSY), which is a model-driven framework for run-time self-architecting and re-architecting of distributed service-oriented software systems [8].

B. Software Adaptation State Machines

An adaptation state machine defines the sequence of states a component goes through from a normal operational state to a quiescent state [2][3]. A component is in the Active state when it is engaged in its normal application computations. A component is in the Passive state when it is not currently engaged in a transaction it initiated, and will not initiate new transactions. A component transitions to the Quiescent state when it is no longer operational and its neighboring components no longer communicate with it. Once quiescent, the component is idle and can be removed from the configuration, so that it can be replaced with a different version of the component. To enable adaptation patterns, as well as the corresponding code that realizes each pattern, to be more reusable, adaptation state machines are encapsulated in software adaptation connectors as discussed next.

C. Software Adaptation Connectors

Software adaptation connectors [3][4] are used to encapsulate adaptation state machines so that adaptation patterns can be more reusable. The adaptation patterns described in this paper include two different types of adaptation connector, *coordinator connector* and *service connector*. The goal of an adaptation connector is to separate the concerns of an individual component (service or coordinator) from its dynamic adaptation. An adaptation connector models the adaptation mechanism for its corresponding service or coordinator. An adaptation connector behaves as a proxy for a component, such that its clients can interact with the connector as if it were the component, as shown in Fig. 1.

IV. HIERARCHICAL SERVICE COORDINATION ADAPTATION PATTERN

In the hierarchical service coordination adaptation pattern for SOA, a higher-level coordinator orchestrates lower-level coordinators, whereas each of the lower-level coordinators is responsible for distributed service coordination. The communication diagram depicted in Fig. 1 shows a general hierarchical coordination pattern where a higher-level parent coordinator coordinates M lower-level child coordinators, each of which interacts with multiple services.

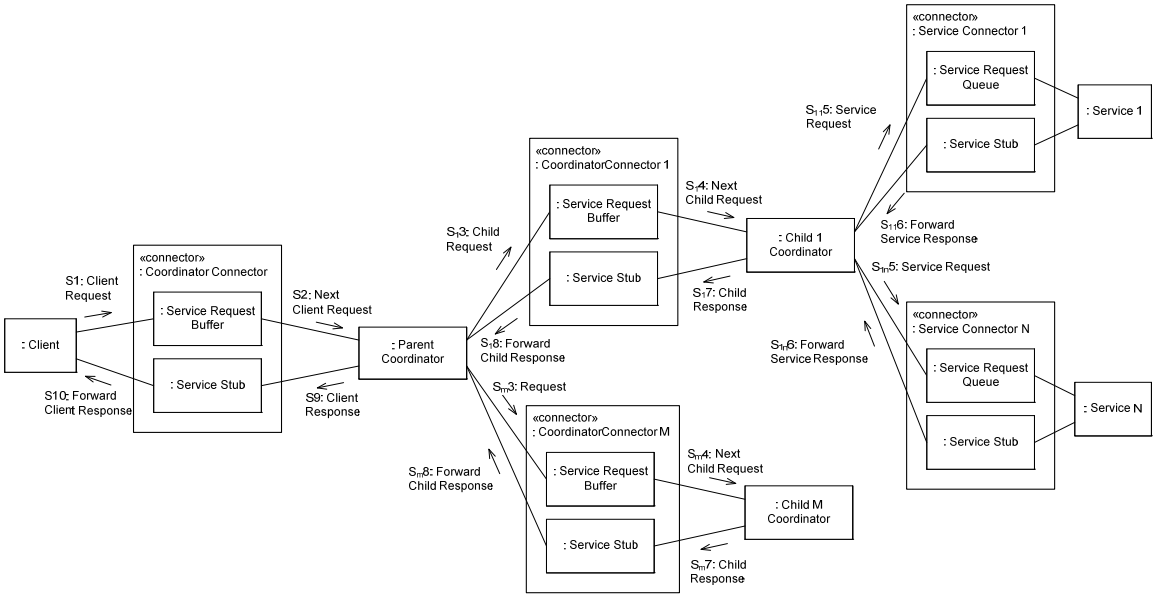


Fig. 1 Hierarchical service coordination communication diagram

An example of hierarchical coordination is a client trip request to the parent coordinator consisting of an airline reservation, a hotel reservation and a car reservation. The parent coordinator decomposes the client request into three smaller requests, which are sent to child coordinators for airline, hotel, and car reservations using a combination of sequential and concurrent coordination (e.g., hotel reservation followed by concurrent hotel and car reservations. Each child coordinator interacts with several individual services (e.g., airline companies) in order to select the most appropriate service. The parent coordinator receives the child coordinator responses and then responds to the client.

The hierarchical service coordination adaptation pattern is organized as follows:

- A parent coordinator is instantiated for each client.
- Two or more child coordinators are instantiated for each parent coordinator.
- A client interacts with a parent coordinator using synchronous message communication; thus, it sends a new request only when it receives a response to its previous request.
- A parent coordinator receives a client request and decomposes it into smaller requests, which are sent to child coordinators. The parent coordinator communicates with the child coordinators either sequentially or concurrently.
- A child coordinator communicates with multiple services sequentially or concurrently. It uses independent service coordination for stateless services [3] and transaction based communication (e.g., two phase commit protocol) for stateful services [4].
- The parent coordinator responds to the client after it has received responses from each of the child coordinators.

To address hierarchical service adaptation it is necessary to consider adaptation of parent coordinators, adaptation of child coordinators, and adaptation of individual services.

A. Dynamic Run-time Adaptation for Hierarchical Coordination

Using the hierarchical service adaptation pattern, the parent coordinator component can be removed or replaced after it has received all the responses from the child coordinators and sent its response to the client. A child coordinator can be removed or replaced after it has received responses from all the services invoked and sent its response to the parent coordinator. On the other hand, a service can be removed or replaced after it completes the current service execution in the case of a sequential service, or after completing the current set of service executions in the case of a concurrent service.

The solution involves one coordinator connector for the parent coordinator and one coordinator connector for each child coordinator, as depicted in Fig. 1. Each connector encapsulates the adaptation state machine for its corresponding coordinator. This is possible because a connector tracks the states of its corresponding coordinator, since it receives (and forwards) each upstream message sent to the coordinator and each downstream message sent by the coordinator.

Figures 2 and 3 depict the adaptation state machines executed by the coordinator connectors for the parent coordinator and the child coordinator respectively. Applying separation of concerns, parent and child coordinators deal with coordination decisions while their corresponding connectors address adaptation decisions. Thus, the parent coordinator connector encapsulates the adaptation state machine of the parent coordinator it communicates with, whereas the parent coordinator interacts with multiple child coordinators via their coordinator connectors.

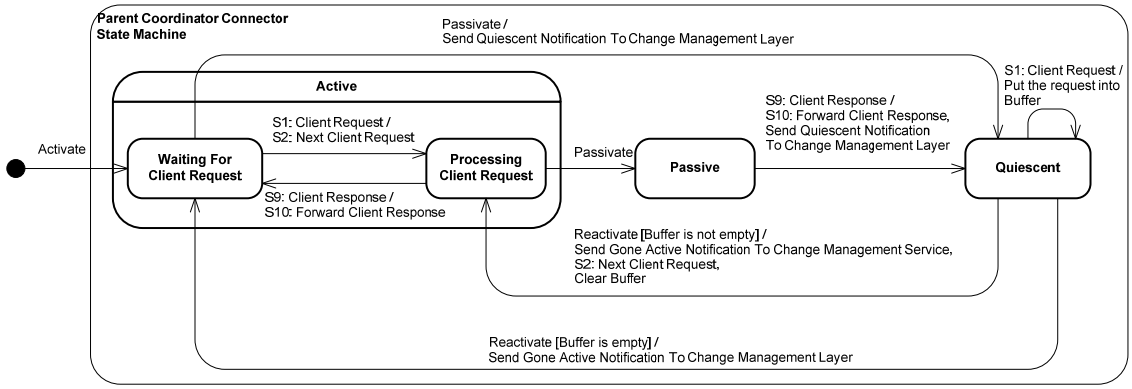


Fig. 2 Parent coordinator adaptation connector state machine

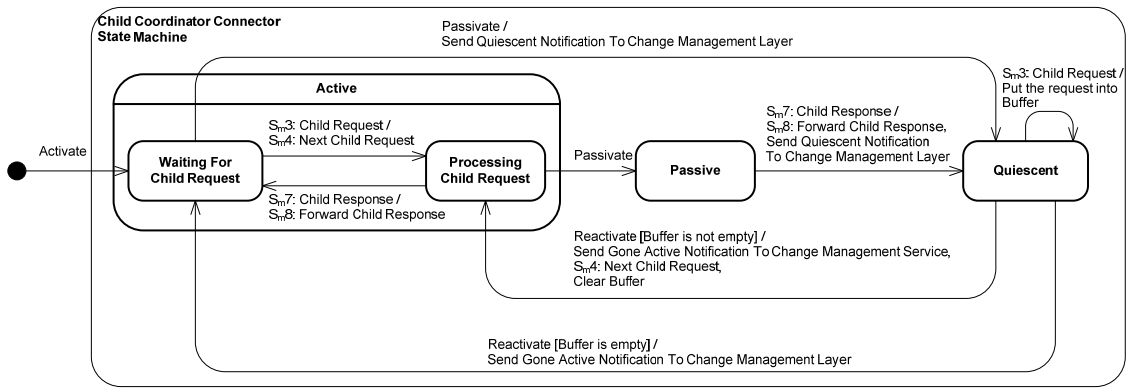


Fig. 3 Child coordinator adaptation connector state machine

B. Adaptation of Parent Coordinator

As described in the previous subsection, the parent coordinator connector encapsulates and executes the adaptation state machine for the parent coordinator, shown in Fig 2. (Because of this, the state names reflect the states of the coordinator and not the connector). There are three main states, Active, Passive, and Quiescent. In the Active state, the coordinator is operating normally and its state machine is in one of the two substates of the composite Active state. As shown in Fig. 2, the parent coordinator connector is initially in Waiting for Client Request substate. When it receives a request from the client (message S1 in Fig 1), the connector transitions to Processing Client Request substate (event S1 in Fig 2) and forwards the next client request to the Parent Coordinator (action S2 in Fig 2 and corresponding outgoing message S2 in Fig 1). The parent coordinator then interacts with the child coordinators. When the parent receives the responses from all its children, it sends the client response (message S9 on Fig.1) to the connector. The parent connector transitions back to Waiting for Client Request state (event S9 on Fig. 2) and forwards the response to the client message (action S10 on Fig.2 and corresponding message S10 on Fig. 1).

To initiate dynamic adaptation of the parent coordinator, a Change Manager (CM) [2][3], which is part of the SASSY adaptation framework (see IIIA and [8]), sends the Passivate command to the parent coordinator connector. If the connector is in the Waiting for Client substate (Fig 2), it transitions directly to the Quiescent state; the action is to send a quiescent notification message to CM. Alternatively, if the connector is in the Processing Client Request substate when it receives a Passivate command, it transitions to the Passive state because the parent coordinator is still interacting with the child coordinators to complete the client request. When the connector receives the Client Response (message S9 on Fig.1) from the Parent Coordinator (indicating that the coordinator has completed the client request), it transitions to Quiescent state (event S9 on Fig. 2). The actions are to forward the response to the client (action S10 on Fig. 2 and message S10 on Fig. 1) and to send a quiescent notification to the CM. In Quiescent state, the parent coordinator is idle and ready to be replaced. If a new client request arrives in Quiescent state, the request is stored in a buffer. After the coordinator has been replaced, CM sends a Reactivate command to the coordinator. If the buffer is empty, the connector transitions to Waiting for Client Request. Otherwise, the connector transitions from Quiescent state to Processing Client Request and sends the buffered client request to the reactivated parent coordinator (action S2 on Fig. 2 and corresponding message on Fig. 1).

C. Adaptation of Child Coordinator

Each child coordinator connector in Fig. 3 encapsulates the state machine for its corresponding child coordinator. It receives child requests from the parent coordinator and forwards these to the child coordinator. The connector receives child responses from the child coordinator and forwards these to the parent coordinator. When the child connector receives a Passivate command from CM, it transitions to Quiescent state (if it is waiting for a client request) or to Passive state (if it is processing a child request). In the latter case, when the connector receives the child response from the child coordinator, it transitions to Quiescent state and forwards the child response to the parent coordinator.

If the child coordinator coordinates stateless services independently, independent coordination adaptation patterns [3] are applied to the adaptation of a service in the hierarchical coordination pattern, as depicted in Fig. 3 and described above. If a child coordinator orchestrates stateful services using a Two-Phase Commit Protocol, the two-phase commit coordination adaptation pattern described in [4] is applied.

D. Adaptation of Services

A concurrent service services multiple client requests concurrently. The adaptation state machine for a concurrent service connector is shown in Fig. 4. The service connector receives service requests from a child coordinator as well as from other clients and forwards them to the service. For a concurrent service, the service can be

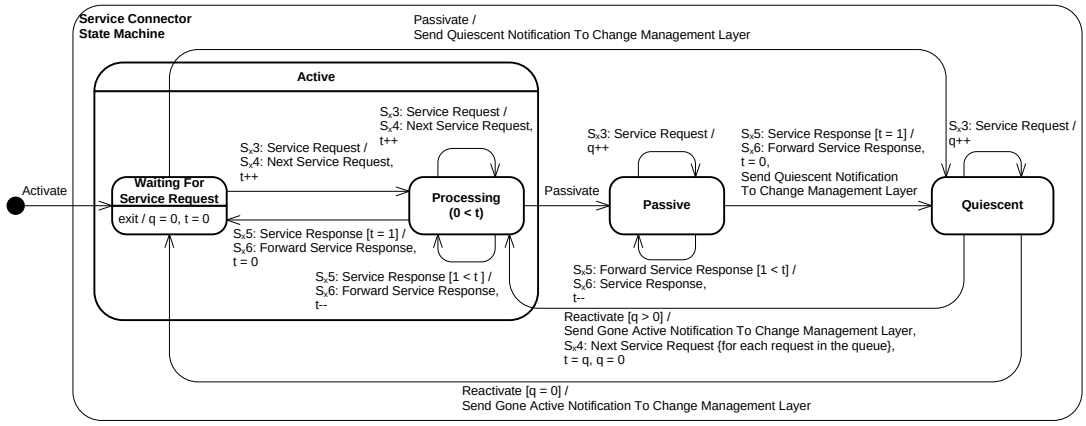


Figure 4 Concurrent service adaptation connector state machine

removed or replaced after it has completed the service requests it has received via the service adaptation connector. The service connector keeps a count t of the requests currently being executed by the service, incrementing the count when a new request is sent to the service and decrementing the count when the response is received and then forwarded to the appropriate client or child coordinator.

If a passivate command is received from CM, the adaptation connector transitions to Passive state if busy, where it waits for the current service requests to complete. New service requests are queued in a service request queue, which is managed by a queue counter q. When the current service requests are completed, the adaptor transitions to Quiescent state. When it receives the reactivate command from CM, the service connector sends the queued service requests to the replacement service and transitions to Processing state.

V. Validation of Hierarchical Service Coordination Adaptation Pattern

The SOA adaptation patterns were validated using the SASSY dynamic run-time software adaptation framework [3][8]. The prototype implementation of the SASSY framework is based on Web services and was developed using open-source SOA frameworks, namely Eclipse Swordfish and Apache CXF. A prototype emergency response system was developed using this framework. Using this framework, validation of a service adaptation pattern consists of executing change management scenarios, performing the run-time adaptation from one configuration to another, and resuming the application after the adaptation.

For the validation of the hierarchical service coordination adaptation pattern, the emergency response system consisted of a region (parent) emergency coordinator that assigned emergency requests to three district (child) emergency coordinators, which each coordinated their local fire engine and ambulance services. Separate adaptation scenarios were executed for the parent and child coordinators and were monitored using execution traces for the parent and child adaptation connectors. The execution trace for the parent coordinator connector is shown in Fig. 5, during which adaptation of the parent coordinator is carried out. The trace depicts the sequence of states the connector transitions through, starting in Idle state. The connector receives a client request, transitions from Idle to Processing state, and sends the new transaction to the parent coordinator. It then receives a Passivate command from CM and transitions to Passive state. When the transaction completed response is received from the parent coordinator, the connector transitions to Quiescent state. In this state, the parent coordinator can be replaced. While in Quiescent state, a new request arrives at the connector from the client and is queued. After adaptation is completed, the connector receives the Reactivate command from CM, transitions to Processing state, and sends the queued request to the new parent coordinator. After the transaction is completed, the connector transitions back to Idle state.

An execution trace for adaptation of a child coordinator is shown in Fig. 6. This scenario shows that child coordinator connector transitions to Processing state after receiving a request from the parent coordinator, which it then sends to the child coordinator. After receiving a Passivate command, the connector transitions to Passive state. When the connector receives the completion message from the child coordinator, it transitions to Quiescent state. In this state, the child coordinator can be adapted. While in Quiescent state, the connector receives a new request, which it queues. After receiving the Reactivate command, the connector then transitions to Processing state and sends the queued request to the child coordinator. When this request is completed, the child coordinator connector transitions to idle state.

In summary, the validation scenarios confirm that the parent and child coordinator adaptation connectors behaved as specified, transitioning from Processing to Passive to Quiescent states and then back to Processing state, while sending and receiving the expected messages.

 ISBN: 978-1-61208-341-4

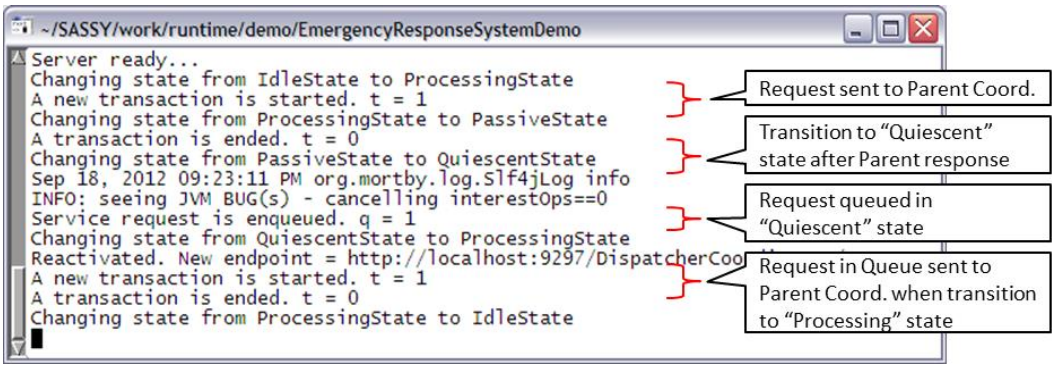


Fig. 5 Execution trace of Parent Coordinator Connector in hierarchical service coordination

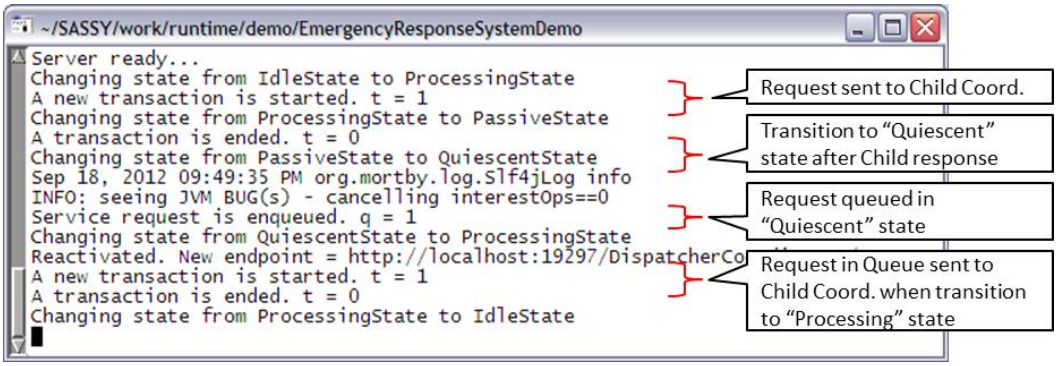


Fig. 6 Execution trace of Child Coordinator Connector in hierarchical service coordination

VI. CONCLUSIONS

This paper has described how software adaptation can be applied to hierarchical coordination in service oriented systems. The main contributions of this paper are:

1. Adaptation pattern for distributed hierarchical service coordination, which can operate with either stateless or stateful services. For hierarchical service coordination with distributed transactions, the pattern corresponds to the compound transaction pattern [6], in which a compound transaction is decomposed into two or more atomic transactions.
2. Design of adaptation connectors for distributed service coordination. Adaptation connectors encapsulate the adaptation state machines for the adaptation pattern to separate the concerns of an individual service or coordinator from software adaptation.

Future work consists of investigating performance issues of dynamic adaptation for service-oriented architectures, developing additional adaptation patterns, and considering recovery from service failure.

ACKNOWLEDGMENTS

This research was partially supported by grant CCF-0820060 from the National Science Foundation. The authors gratefully acknowledge the contributions of D. Menasce, S. Malek, J. Sousa, N. Esfahani, and J. Ewing to the SASSY project.

REFERENCES

[1] J. Kramer and J. Magee, "The Evolving Philosophers Problem: Dynamic Change Management", IEEE Transactions on Software Eng., Vol. 16, No. 11, 1990, pp. 1293-1306.
[2] H. Gomaa, "A Software Modeling Odyssey: Designing Evolutionary Architecture-centric Real-Time Systems and Product Lines", Springer Verlag LNCS 4199, 2006, pp 1-15.
[3] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D. Menasce "Software Adaptation Patterns for Service-Oriented Architectures", Proc. ACM Symp. on Applied Computing, March 2010, pp. 462-469, Sierre, Switzerland.
[4] H. Gomaa and K. Hashimoto, "Dynamic Self-Adaptation for Distributed Service-Oriented Transactions", Proc. SEAMS Symposium, Zurich, Switzerland, June 2012, pp. 12-20.
[5] A. J. Ramirez and B. H. Cheng, "Applying Adaptation Design Patterns," Proc. 6th Intl. Conf. on Autonomic Computing (ICAC), Jun. 2009, pp. 69-70.
[6] H. Gomaa and K. Hashimoto, "Dynamic Software Adaptation for Service-Oriented Product Lines", in Proc. Intl Wkshp on Dynamic Software Product Lines, Munich, Germany, August 2011.
[7] H. Gomaa, "Software Modeling and Design", Cambridge University Press, 2011.
[8] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, SASSY: A Framework for Self-Architecting Service-Oriented Systems", IEEE Software, Vol. 28, No. 6, 2011, pp. 78-85.
[9] G. Li, et al., "Facilitating Dynamic Service Compositions by Adaptable Service Connectors", International Journal of Web Services Research, Vol. 3, No. 1, 2006, pp. 67-83.
[10] F. Irmert, T. Fischer, and K. Meyer-Wegener, "Runtime adaptation in a service-oriented component model", Proc. SEAMS Symposium, May 2008, pp. 97-104.
[11] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge", Proc Intl. Conference on Software Engineering, Minneapolis, MN, May 2007, pp. 259-268.

 ISBN: 978-1-61208-341-4

Towards a More Rigorous Foundation of Complex Adaptive Systems in Management Science: Dealing with Misnomers and Metaphors

Leslie Klieb[1, 2] , Merle Rhoades[1, 3], Bill McKelvey [4]
[1] University of Liverpool in partnership with Laureate Education, Inc., Liverpool, UK
[2] Webster University Thailand, Bangkok, Thailand
[3] East Colorado SBDC and U. of Northern Colorado, Greeley, CO, USA
[4] Kedge Business School, Marseille, France
{leslieklieb@gmail.com, bbr@q.com, mckelveybill1@gmail.com}

***Abstract*—This paper provides a framework for organizational Complex Adaptive Systems without directly referring to biology (evolution theory), physics and mathematics. The model can reproduce most of standard management science and sheds light on new ideas in corporate strategy. Possibilities for numerical simulation are discussed.**

Keywords-adaptive systems; management science; agents; modelling.

I. INTRODUCTION

The existing literature about organizations seen through a Complex Adaptive Systems (CAS) lens (Lewin, 1992 [1]; Kauffman, 1993 [2]; Holland, 1995 [3]; Maguire et al., 2006 [4]) usually borrows for a discussion of system properties from biology, mainly from evolutionary dynamics (Holland, 1975 [5]; Mandelbrot, 1982 [6]; Nicolis and Prigogine, 1989 [7]; Kauffman, 1993 [2]; Aldrich, 1999 [8]; McKelvey, 1982 [9]; Nelson and Winter, 1982 [10]) and from physics (Prigogine, 1955 [11]; Kaye, 1993 [12]; Cramer, 1993 [13]; Gell-Mann, 1994 [14]). Not only does this make this the literature extremely hard to read for managerial scientists and practitioners, it also obscures that CAS in social science have essential differences with biological systems. We show here that organizational systems (businesses, but not exclusively) can be easily interpreted as a collection of agents (Carley, 1992 [15], 1999 [16]; Carley and Hill, 2001 [17]). While standard CAS literature avoids specifying the attributes of the agents and their interactions (schemata), we show here that specifying these schemata leads to an easily understandable framework that encompasses most of standard business science, and gives a clear interpretation of most standard CAS literature on organizations. It also makes it easier to see what the differences are relative to the evolution of biological systems (McKelvey, 1982 [9]; 1994[18]; Maguire et al, 2006 [4]; McKelvey et al., 2013 [19]).

The plan of this work is as follows. In Section II, a brief overview of some salient literature is given. In section III, schemata between agents are formulated. After that, the rest of the paper is devoted to how these schemata, although admittedly (too) simple to be predictive of all behavior of agents, are sufficient to reproduce most standard CAS management literature. So, in Section IV, sudden shifts in phase space from punctuated equilibrium are discussed. Section V focuses on the Edge of Chaos. Section VI focuses on how the agent description reduces to many standard management science descriptions when certain interactions are small compared with others and therefore can be neglected. As these standard management descriptions have been usually experimentally verified, this provides the necessary link with empirical descriptions in a large number of limiting cases. Section VII provides an interpretation of the Soft System Methodology in an agent view. Section VIII does the same for Action Research. Section IX provides some conclusions and an outlook.

II. LITERATURE REVIEW FOR CAS

Especially when the application of CAS to business was developed, authors frequently considered the business ecology as analogous to evolutionary biological systems (for instance Kauffman, 1993 [2]; Bak, 1996 [20]; Anderson, 1999 [21], Gell-Mann, 1994 [14]). This has often been very fruitful. McKelvey (1982, 1994) [9][18], McKelvey et al. (2013) [19], and in an unpublished work McKelvey (2002) [22] identified many patterns that are common between the dynamics of biological systems under influence of evolutionary forces and the dynamics of business systems. However, the relationship between biological and social CAS remains unclear. In this work, we show:

- Both may be represented and studied via agent-based (computational) models of system dynamics (aggregates of agents)
- In general, models need to be sufficiently detailed so that they can reproduce the characteristics of dynamics. On the other hand, models should focus on essentials and not be cluttered with too many details. Davis and Eisenhardt (2007) [23] called this the “sweet spot” of model design. We give arguments here that it is possible to develop successful models of both biological and business dynamics independently. Such independent modeling has already been done successfully for biology, and it will be done in a heuristic way in this work for the kinds of CAS systems in human organizations. Such a model for social organizations

 ISBN: 978-1-61208-341-4

can reproduce most of business science, and is, therefore, empirically valid. A useful beginning is the model developed by Carley and Hill (2001) [17].

- There is a large similarity between the two models, and this explains similarities in evolutionary dynamics, like the patterns found by McKelvey (2002) [22].

- There are also essential differences between the two models, and this explains where the analogies break down. Biology is a metaphor, not an explanation for business science (McKelvey et al., 2013 [19]).

III. THE MODEL

As postulated by an agent-model of CAS that applies to most of management science, agents have the following attributes—mostly developed early on by Carley et al. (e.g., Carley, 1992 [15], 1999 [16], 2002 [24]; Carley and Svoboda, 1996 [25]; Carley and Hill, 2001[17]):

- Needs for food, energy, shelter, and similar (equivalent needs for organizations)
- Need company and bonding mediated by connections (attractive force)
- Need space (privacy, physical room (repellent force)
- Needs are hierarchical, a lower more essential need can overcome a higher need, analogous to Maslow's hierarchy.
- Intentionality (considered also as a need, for the ease of discussion here)
- Agents can (and actually like) to learn (learning = behavior change under influence of stimuli on longer time scales).
- Agents cannot have perfect knowledge about other agents. There is always interpretation.

Satisfying needs comes with costs. The number of needs is countable, possibly (and probably) infinite. Agents in this context can be people but also organizations, and every other kind of system that is studied in social science. We will show that the above needs are sufficient to define a CAS. The system shows many characteristics of CAS as discussed in the management literature and provides helpful guidance for managers in understanding system effects.

Agents try to do what they perceive as best for them to fulfil their needs and survive while taking into account the costs to do so, and therefore, make an assessment of their needs and situation and try to improve their situation. CAS literature calls this measure of how well needs are satisfied (fitness), but it is a perception of 'fitness' (if fitness is taken as defined in biology by ability for survival). Perceived fitness is an assessment how well various needs are satisfied and in which direction an agent would like to move in order to satisfy better the needs and enhance survival or other measures of success. This includes an outlook for the future. Perceived utility is the gain that can be made in perceived fitness (a small extension of or maybe identification with the economic term). Utility is a function of the needs.

A technical assumption is based on transitivity of choices: If Choice A is preferred over B and also B is preferred over C, then A is preferred over C. In this case, perceived fitness can be measured on a one-dimensional scale, it is a mathematical non-linear function of all variables/needs.

Given these assumptions, agents always have a perception of a best course of action. This is sufficient to define a fitness landscape and a phase space consisting of {needs x perceived fitness}. The dynamics of agents are determined by their attempts to increase perceived fitness. Their interactions lead to a CAS: the dynamics are irreducible (cannot be compartmentalized). Agents form systems because of the long-range attraction and short-range repulsion, which leads to an optimal size with respect to costs. An example provides the work by Bettencourt, 2013 [26] on the size of cities or the work of Krugman, 1996 [27] on spatial economy. The schemata also cause the system to operate far from equilibrium (Lewin, 1992 [1]; Cramer, 1993 [13]). The above needs/schemata are insufficient to explain all observed dynamics. For instance, for human agents, psychological factors (in principle, part of the schemata) are not included in the above model. In modern times, dynamics between agents are mostly determined by other agents and not by a non-human environment (like forces of nature). The interaction between agents is now termed co-evolution (Kauffman, 1993 [2]).

In a business science context, agents are heterogeneous; they can be and usually are different. These differences are expressed in that they react differently to their environment and to other agents (because of differences in attributes), and that therefore they tend to have different interactions. Technically, agents have different schemata (Carley, 1992 [15], 1999 [15]; Carley and Hill, 2001 [17]; Ilgen and Hulin, 2000 [28]).

IV. PUNCTUATED EQUILIBRIUM

Most agents are close to a local peak, but not to a more optimal but (usually) more distant global peak (Carley and Svoboda, 1996 [25]). Usually, most changes in an agent's environment can be accommodated by making gradual moves. Changes in environment can be limited to changes in the relative height of peaks. Sometimes peaks disappear or new peaks emerge. Such events can have a dramatic influence, because disappearance or growth of one peak can lead to a domino effect and influence peaks in the neighborhood of those peaks that in turn influence peaks in their neighborhood and so on, leading to a major configuration (Barabási, 2005 [29]). This can lead to a huge change in the fitness landscape for an agent. This shows that often changes for an agent can be accommodated slowly, close to a dynamic equilibrium, but sometimes the fitness landscape, and with that the dynamics of an agent, changes tremendously and no slow (adiabatic) change is possible anymore. Equilibrium is punctuated by sudden disruptions (Bak and Sneppen, 1993 [30]; Romanelli and Tushman,

 ISBN: 978-1-61208-341-4

1994 [31]; Bak, 1996 [20]; Gould and Eldredge, 2000 [32]). Disruptive technology is an example of this (Andriani and Cohen, 2013 [33]).

V. THE EDGE OF CHAOS

In every assembly of such agents, intuitively, there are three regions: not enough meaningful interaction between them to speak of a system, enough interaction so that tacit and explicit knowledge is exchanged between nodes, and a region in which too many interactions make the system uncontrollable and overreacting (Langton, 1990 [34]; Kauffman and Johnson, 1991 [35]; Lewin, 1992 [1]; Brown and Eisenhardt, 1998 [36]; Pascale, Millemann, and Gioja, 1999 [37]). The transition between "enough interaction" and "too much" is metaphorically called the edge of chaos. Although in certain mathematical models in biology the uncontrollable region is chaotic in the mathematical sense, here the dynamics is not predictable enough to make such general mathematical statements as the existence and size of Lyapunov coefficients (Montroll and Badger, 1974 [38]).

In organizations, overly connected regions of the CAS that have too few links with their environment are called silos (LaBonte, 2001 [39]; Diamond, Stein, and Allcorn, 2002 [40]; Dell, 2005 [41]). They are a sad consequence of the heterogeneity of agents, which in good cases makes the system more adaptive. If agents were homogeneous, exactly similar, each agent would have the same type of interactions with other agents, and the phase diagram would still have symmetry breaking, but not on such a large scale. It would be homogenous throughout each of the three regions. Therefore, in an organization the three-region model is simplistic. Regions of more and less connections are scattered all over the organization (often department-wise, or otherwise as informal groups, see above for arguments how such more or less stable subsystems diminish costs). It does not help that because of fractality (i.e., self-similarity), these subsystems have their own edge of chaos (Schroeder, 1991 [42]). A silo is an uncontrollable region where link inside link density is too high from the point of view of controllability by the enveloping organization. Intuitively, it is similar to a type of attractor (fixed point, limit cycle, limit torus, strange attractor, not necessarily chaotic).

VI. LINKS WITH KNOWN MANAGEMENT SCIENCE

The agent model given here reproduces a large number of disparate management fields of study. It reproduces most of the standard CAS literature. It deviates where the assumptions are different, for instance Stacey's (2011) [43] theory of responsive processes stresses very different interactions between agents (different schemata). This makes the scope of applicability of Stacey's theories very different. In fact, there is increasing evidence that various kinds of both static and dynamic aspects of organizations are self-similar from small to large to environmental scales, Batty and Longley (2004) [44], Newman (2005) [45], Andriani and McKelvey, (2007 [46], 2009 [47]), McKelvey and Salmador (2011) [48], and McKelvey, Lichtenstein and Andriani (2013) [49] offer 200+ examples of how the many variables characterizing organizations result in fractal (i.e., Pareto long tailed rather than normal) distributions.

A. The agent in its environment and misalignment issues: static descriptions

Organizational CASs are fractal systems, they exhibit self-similarity in their dynamics, and because of this, similar social structures arise at various sizes (Stanley et al., 1996 [50]; Solé, 2001 [51]; Andriani and McKelvey, 2007 [46], 2009 [47]). Agents are part of many groups of different sizes. All these groups have their own perception of fitness. These perceptions are in general not aligned. The result is that an agent in a group may feel misalignment up to a certain degree, between its own perception and the perceptions of fitness (mission, goal, purpose) of the group to which it belongs. Examples:

Resistance to change: An agent's perception of its own fitness clashes with the perception of the fitness of a group it belongs to. This is usually its employer or boss, but can be a religious or political or other organization.

Principal Agent Problem: Aided by asymmetric information, perception of fitness of a C-level director is misaligned with the perception of fitness of the owners of the firm (who are after maximization of shareholder profit). Note: The existence of asymmetric information comes from the postulate in the schemata that no objective knowledge is possible.

Turnover: an agent feels so much misalignment that it is leaving its group (examples are in employment, marriage, club membership, etc.)

Cognitive dissonance: An agent tries to reconcile misalignment between its own perception of fitness with the group's perception of fitness (Festinger, 1947 [53]). Values held by agents can be understood as the agent's ideas about best direction to go, so these values are part of utility in this scheme.

Ajzen's (2011) theory of planned behavior [52] recognizes environment, i.e., the interactions that one agent feels from other ones, via the subjective norm and shows that this influences the dynamics (intentions leading to behaviors).

Marketing: People do not always go for the least expensive purchase, because buying upscale signals to others their ability to survive (analogous to the potlatch). Giving of presents serves the same purpose.

Global Controller: Holland, 1988 [54] notes that in biological CASs there is no global controller, i.e., no paid boss – even the queen bee doesn't get paid to tell worker bees what to do. However, all organizations have a CEO who is paid to take charge, take control, etc., and lower-level managers who are also paid to be in charge. This asymmetry between agents is probably the most fundamental difference between biological species and herds vs. human organizations.

 ISBN: 978-1-61208-341-4

Theories of leadership: Complex Leadership Theory (CLT). Continuing where Holland (1988) [54] left off, Uhl-Bien et al. (2007) [55] point to the unavoidable consequences of fractality and heterogeneity. In every group, (subsystem), leaders and followers will emerge, because agents are heterogeneous and interactions are asymmetric. Some groups are labelled "formal" and others "informal" but that is pure convention. Leaders in formal group are called administrative leaders and function differently towards the environment and are usually recognized by it. Leadership in informal groups ("adaptive leadership") is often not recognized outside the group. In this framework, it becomes very hard to evaluate objectively people's contributions. Leaders of one group can enhance their fitness and the fitness of their own group sometimes by co-opting the leaders that spontaneously emerged from a different group. This process is called enabling leadership.

Leadership Exchange Theory: All leadership occurs in the space between agents. Theories like Leadership Exchange Theory amount to a more precise specification of the schemata.

Resource-based view of the firm applies to all groups (systems). There is always an advantage in pooling resources from the postulate of bonding.

Test particle approach: Introduce one agent into an organization – i.e., an agent is hired. In a first approximation, the agent's dynamics starts to be determined by the interaction of its own perception of fitness and the influence of all the other agents. This influence of all the other agents on a single agent is called organizational culture. In a second approximation, one can "calculate" the influence that this particular agent's new dynamics (which includes its own previous learning, experience and other attributes) is having on the organization. Then one can "calculate" again the influence of the new organizational dynamics on the person, and so finally arrive at a self-consistent description (in theory, not in practice). The second approximation, the influence of the agent on the culture of its group, is alternatively called leadership, art, volunteerism, and any other way agent influences on a system to which it belongs are named.

All the above aspects have in common that they mirror standard areas of business science. However, in the conventional treatment these normally disparate areas are not put into one unified framework. This shows that the schemata used in this description are powerful enough (Cramer, 1993 [13]) to reproduce standard theory (or alternatively, if you want that interpretation, that many management theories have very simple assumptions about the interactions of the agents) (Williamson, 1975 [56]; Read, 1990 [57]). However, in the above applications they do not really test the dynamics of the system.

B. The agent in its environment: dynamic descriptions

Dynamic capabilities: Many benefits of groups result from pooled resources. This is the resource-based view of the firm (Barney, 1991 [58]; Barney, Wright, and Ketchen, 2001 [59]) (which applies in this view to every CAS, as there is no fundamental difference between a firm and any other CAS). So, this theory is really the resource-based view of the group or system, and results from the nature of the fundamental interactions between agents. When it is necessary for the CAS to increase its fitness because of external events (threats, opportunities), often its resources need to be re-configured. This will need to be done in different ways depending on the amount of turbulence and change. Eisenhardt and Martin, 2000 [60], discern high-velocity and medium-low velocity markets. Under high turbulence, many tools, like standard strategic forecasts, lose their value.

Strategy: Depends on the ability to make a moderately successful prediction of the future of the group where one belongs. Events that can be classified as "punctuated equilibrium" or "black swans" (Taleb, 2007 [61]) are inherently nearly impossible to predict accurately ("black swans" result from the fat tails of power laws; the descriptions of the domino theory of punctuated equilibrium and power laws are probably related). Under moderate turbulence, some prediction might be possible (Eisenhardt and Martin, 19[60]. In the transition from low turbulence to high turbulence regions, a prediction about future changes in the fitness landscape, and therefore strategy, becomes more and more unreliable (Holland, 1995 [3]; Krugman, 1996 [27]; Dooley and Van de Ven, 1999 [62]; Sornette, 2003 [63]; De Vany, 2004 [64]; Sornette et al., 2004 [65]; Baum and McKelvey, 2006 [66]). One of the CAS alternatives is to strengthen connections and upgrade the knowledge of the agents (change their schemata by learning), which moves the organizational culture of the company closer to the "edge of chaos" (Carley, 1999 [16]; Pascale, Millemann, and Gioja, 1999 [37]). The organization is more adaptive and better learning at this point, and this gives it more of a chance to survive as a group (Carley and Hill, 2001[17]). If it fails to do this, its constituent agents will move on to different groups (given enough employment possibilities) and add variety to their new group, as discussed above.

Some economic models rely on heterogeneity of agents and mirror such conclusions, like the work by Melitz (2003) [67].

In the foregoing description, organizational failure can be beneficial because it releases agents to other groups that are hopefully better equipped at this juncture in time. However, it follows also that each organization fails because of some specific circumstances in its ecology (e.g., Blackberry) and (in general) not from some type of generalized low capacity for success (e.g., UK public rail system; Cyprus banks). Survival does not signify a generalized better "health". A bank that survived a financial crisis can still be defenseless against fraud. A software company that was very successful in developing operating systems for PCs might still stumble in with tablets or smartphones (e.g., Blackberry and Nokia). There is a large path dependency here. The amount of control that a CAS has in determining its own future when multiplicative interaction (connectivity) effects instigate extreme events is much more problematic, if not actually

reduced (Anderson, 2006 [68]; McKelvey and Andriani, 2010 [69]; Andriani and McKelvey, 2011 [70]).

In biology, there is no control at all. Survival is random from accidental ability to survive certain threats. The control among human agents comes from their intentionality (we do not want to enter into a discussion if this is real or just an illusion, it makes no difference for this discussion.) Jack Welch, former CEO of GE, is a good example of CEO who created tensions to motivate managers and employees to seek better solutions by changing their objectives and learning from other executives and/or employees (often newly acquired by "M&A" activities), along with various additional complexity elements so as to get employees, departments, divisions, and companies operation closer to the edge of chaos (McKelvey, 2010 [71]).

Computational Simulation [agent-based computational models (ABMs)]: ABMs allow computational simulations when details of the schemata are sufficiently specified. Many models that can be analytically analyzed have chaotic regions [caused by too many connected variables (degrees of freedom)] in the phase space—like the "melting zone" (the Region of Emergent new Order between the Edge of Order and the Edge of Chaos) in Kauffman's (1993) NK-model [2]. Mathematical optimization models work well below the Edge of Order (in the Region of Order). However, instabilities are expected once the system being modeled tips over the Edge of Chaos.(Canuto et al., 2005 [72]; Bruun, 2006 [73]). Averaging over coordinates of the phase space that are judged irrelevant (coarse graining) reduces the degrees of freedom and makes optimization models more feasible. Incorporating feedback mechanisms (intermediate changes in the schemata made by the agents), and other smoothing mechanisms can handle numerical instabilities that are otherwise unavoidable in chaotic regions, which is to say, get the system out of chaos and back into the Region of Emergence.

In realistic ABM simulations, one would also attach probabilities to some of the options that an agent has, because one could not be sure what an agent would do, given the imperfect knowledge an agent has about other agents. This would also require an ensemble-averaging by making many simulation runs (usually somewhere between 250 and 10,000 runs of the same ABM design to get the average). ABM simulations allow the exploration of interesting areas of phase spaces that current management theories do not probe. For instance, does cognitive dissonance play a role in principal-agent issues? ABM simulations make it possible to formulate hypotheses that can be empirically tested and go beyond the over-simplified math-based optimization models that characterize standard management science by making less rigorous simplifications.

VII. CHECKLAND'S SOFT SYSTEM METHODOLOGY

The Soft System Methodology (SSM) of Checkland (2000) [74] and co-workers can be understood as an attempt to transfer diagnostic tools from "hard systems" as much as possible to "soft systems". Hard systems are those that can be observed from the outside and the dynamics measured with arbitrary precision limited by technology or physics. Hard systems are diagnosed with instruments via observations. Such observations provide a snapshot in time about the system. Experiments can probe its dynamics by disturbing the system.

We assume that there is one (or in any case very few) observers in an organizational CAS who want to know system-wide properties. Most agents will be satisfied with local observations because their dynamics are more determined by these. Others do not have the access or the tools, or do not have the impetus. Managers, who are the administrative leaders in the formal organizations, usually make such more system-wide observations because they need to confront "messy" or "wicked" problems that do not have a "best" solution. At best, managers can develop "approximate" solutions, which may be improved over time—usually in changing environments in which no single, permanent solution is possible, relevant or desirable.

Some of their diagnostic tools are:

- Agents' own observations of the system dynamics: This entails a shift to an interpretive stance, as the observing agent has usually no means to validate its observations in an objective way. (This relates to the postulate in the schemata that no objective knowledge is possible for an agent about another agent)
- Possibly objective observations like business statistics, stored computer records, and so on. These data usually require interpretation as well.
- Ask other agents for their observations. The Soft Systems literature calls this "collecting worldviews". Diverging worldviews are a hallmark of a messy problem. Such messy problems are typical for open systems, because these provide the adaptive tensions that create such "messy" problems.

In principle, the Checkland's soft tools could be applied to a larger organization, but in practice, they do not scale up sufficiently—they are overwhelmed by too many degrees of freedom. In a small subsystem, however, Checkland's SSM approach may offer different worldviews than can shed light on smaller scale messy problems.

The insider/outsider problem boils down to the impossibility for the manager-agent to hold the two measurements resulting from using different diagnostic tools in its mind at the same time. It is not fruitful to consider this as a deep epistemological problem as is sometimes done in the literature. It is just observing the system from two different positions. There is no mystery in that the two views do not coalesce and that looking at a system from two

different points of view with different diagnostic tools does not give a consistent description.

The process of collecting worldviews and possibly get to some convergence among stakeholders, amount to a snapshot and does not lead to new knowledge about the dynamic properties of the system. However, it is more cognizant of systemic issues than most other business and organizational science. On the one hand, an ABM allows the mixing of different views in different contexts to search for the best-at-the-time perspective. On the other, the ABM allows a manager to search for the parts of systemic issues that are essentially the same across the system vs. those that are demonstrably different.

VIII. ACTION RESEARCH

The only way to learn something about the dynamics of a system for which there is no mathematical model is to look at the effects under disturbances. Such disturbances can come from the environment. Much research has been done in observing shocks to systems; most case studies fall in this category. Only via experiments can one alter the disturbances affecting an organization. But we can't put organizations into laboratories. ABMs, however, allow to simulate organizational phenomena and then conduct simulated experiments.

Alternatively, manager-agents can sometimes apply more controlled shocks themselves. This provides an interpretation in CAS terms of the work of Lewin (1946) [75]. Applying shocks and studying scientifically the resulting changed dynamics is, in this interpretation, Action Research. Managers can apply Actions themselves, but it makes sense to first learn as much of the system as possible. One tool is SSM.The problem with SSM is that the static snapshot is little predictive about the dynamics, and so can lead to unintended and unforeseen consequences. But again, doing this in real time with real people could have negative consequences. Safer to use an ABM.

This provides a useful demarcation for what should be called Action Research and what not. Action Research is the scientific study of the dynamical properties of systems by applying shocks in a controlled way and studying the results in an accepted (quantitative or qualitative) way. This criterion, compatible with Checkland's, is very different from Coghlan's [76], for instance. One of the most important points of difference is that our and Checkland's research see CAS and Action Research as (descriptive) science and not as a tool for emancipation or other ethically driven goals. Such goals are possible and compatible, but they are not part of a scientific description. As an alternative for direct observation, this one can do with ABMs.

IX. CONCLUSION AND OUTLOOK

We have shown that it is possible to give a transparent account of CAS with human agents as the indivisible smallest elements that account for most of the characteristics of organizations as they are discussed in management science. This clarifies the relationship between biological CAS systems and organizational ones.

Similarities as well as differences between the models are very important.

- Business agents are inherently less homogeneous than the agents in biology, making fractality much more prominent in business systems. Business agents are constantly adjusting their behavior over a much larger range than in biology, where phenotype behavior is generally set by genotype. Consequently:

- Dynamics is less predictable in business system because of the many degrees of freedom. ABMs become the more relevant method since they offer modeling options and results across a much broader range of interaction effects and nonlinear dynamics resulting from connectivities among some number of heterogeneous agents. Math models cannot be successfully applied to such phenomena.

- Timescales are much smaller. Biological evolution plays out over hundreds of generations. Businesses change at a scale within the lifetime of many organizations, and business adapt. This is possible because of much faster learning in human than in most biological systems, where most evolutionary change is due more to the genetic structure of offspring than the learning abilities of living phenotypes (Darwin, 1859 [77]), though many biologists now place some emphasis on "organic learning" (i.e., learning and change during a phenotype's lifetime (Baldwin, 1896 [78]; Simpson, 1953 [79]; Crispo, 2007 [80]; Badyaev, 2009 [81]; Kauffman, 2013 [82]; Scarfe, 2013 [83]). Survival in changing environments is a function of learning quickly as needed in addition to surviving because of genetic, structural or endemic advantage.

REFERENCES

[1] R. Lewin, Complexity: Life at the Edge of Chaos, University of Chicago Press, Chicago, IL, 1992 [2nd ed. 1999].

[2] S. A. Kauffman, The Origins of Order. Oxford, UK: Oxford University Press, 1993.

[3] J. H. Holland, Hidden Order: How Adaptation Builds Complexity. Reading, MA: Addison Wesley, 1995.

[4] S. Maguire, B. McKelvey, L. Mirabeau, and N. Öztas, "Organizational complexity science," in S. R. Clegg, C. Hardy, T. Lawrence, W. Nord (Eds.), Handbook of Organizational Studies, 2nd ed. Thousand Oaks, CA: Sage, pp. 165–214, 2006.

[5] J. H. Holland, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. Ann Arbor: University of Michigan Press, 1975.

[6] B. B. Mandelbrot, The fractal geometry of nature. New York: Freeman, 1982.

[7] G. Nicolis, I. Prigogine, Exploring Complexity: An Introduction. New York: Freeman, 1982.

[8] H. E. Aldrich, Organizations Evolving. Thousand Oaks, CA: Sage, 1999.

[9] B. McKelvey, Organizational systematics: Taxonomy, evolution, classification. Berkeley, CA: UC Press, 1982.

 ISBN: 978-1-61208-341-4

[10] R. R. Nelson, and S. G. Winter, An Evolutionary Theory of Economic Change. Cambridge, MA: Harvard University Press, 1982.

[11] I. Prigogine, An Introduction to Thermodynamics of irreversible Processes. Springfield, IL: Thomas, 1955.

[12] B. Kaye, Chaos and Complexity. New York: VCH, 1993.

[13] F. Cramer, Chaos and Order, (trans. D. L. Loewus). New York: VCH, 1993.

[14] M. Gell-Mann, The quark and the jaguar. New York, NY: Freeman, 1994.

[15] K. M. Carley, "Organizational learning and personnel turnover," Organization Science, vol. 3, no. 1, pp. 2-46, 1992.

[16] K. M. Carley, "Learning within and among organizations." In: P. C. Anderson, J. A. C. Baum, A. S. Miner, (Eds.), Advances in Strategic Management, vol. 16. Elsevier, New York, pp. 33-56, 1999.

[17] K. M. Carley and V. Hill, "Structural change and learning within organizations," In: Lomi, A., and Larsen, E. R. (Eds.), Dynamics of Organizations: Computational Modeling and Organizational Theories, Cambridge, MA. MIT Press, pp. 63-92, 2001.

[18] B. McKelvey, "Evolution and organization science," in J. A. C. Baum and J. V. Singh (Eds.), Evolutionary dynamics of organizations. New York: Oxford, pp. 314–326, 1994.

[19] B. McKelvey, M. Li, H. Xu, and R. Vidgen, "Re-thinking Kauffman's NK fitness landscape: From artifact & groupthink to weak-tie effects." Human Systems Management, vol. 32, no. 2, pp. 17-42, 2013.

[20] P. Bak, How nature works: the science of self-organized criticality. New York, NY: Copernicus, 1996.

[21] P. Anderson, Perspective: Complexity theory and organization science. Organization Science, vol. 10, no. 3, pp. 216-232, 1999.

[22] B. McKelvey, "Managing coevolutionary dynamics," Working paper, UCLA Anderson School of Management, Los Angeles, CA, 2002.

[23] J. P. Davis and K. M. Eisenhardt, "Developing theory through simulation methods," Academy of Management Review, vol. 32, no. 2, pp. 480-499, 2007.

[24] K. M. Carley, "Simulating society: The tension between transparency and veridicality," Proc. Workshop on Social Agents: Ecology, Exchange, and Evolution. Chicago, IL: University of Chicago, 2002.

[25] K. M. Carley and D. M. Svoboda, "Modeling organizational adaptation as a simulated annealing process," Sociological Methods and Research, vol. 25, no. 1, pp. 138-168, 1996.

[26] L.M.A. Bettencourt, "The origins of scaling in cities," Science, vol. 340, no. 6139, pp. 1438-1441, 2013.

[27] P. Krugman, The self-organizing economy, Malden, MA: Blackwell, 1996.

[28] D. Ilgen, and C. Hulin, (Eds.) Computational Modeling of Behavior, in Organizations. Washington, DC: American Psychological Association, 2000.

[29] A.-L. Barabási, "The origin of bursts and heavy tails in human dynamics." Nature, vol. 435, no. 7039, pp. 207-211, 2005.

[30] P. Bak, and K. Sneppen, "Punctuated equilibrium and criticality in a simple model of evolution." Physical Review letters, vol. 71, no. 24, pp. 4083-4086, 1993.

[31] E. Romanelli, and M.L. Tushman, "Organizational transformation as punctuated equilibrium: An empirical test." Academy of Management, vol. 37, no. 5, pp. 1141-1166, 1994.

[32] S.J. Gould and N. Eldredge, "Punctuated equilibrium comes of age," Nature, vol. 366, no. 6452, pp. 223-227, 1993.

[33] P. Andriani and J. Cohen, "From exaptation to radical niche construction in biological and technological complex systems," Complexity, vol. 18, no. 5, pp. 7-14, 2013.

[34] C. G. Langton, "Computation at the edge of chaos: Phase transitions and emergent computation," Physica D: Nonlinear Phenomena, vol. 42, no. 1, pp. 12-37, 1990.

[35] S. A. Kauffman and S. Johnson, "Coevolution to the edge of chaos: Coupled fitness landscapes, poised states, and coevolutionary avalanches," Journal of Theoretical Biology, vol. 149, no. 4, pp. 467-505, 1991.

[36] S. L. Brown, and K.M. Eisenhardt, "Competing on the edge; Strategy as structured chaos," Boston, MA: Harvard Business School Press, 1998.

[37] R. T. Pascale, M. Millemann, and L. Gioja, Surfing the Edge of Chaos. New York, NY: Random House,1999.

[38] E. W. Montroll and W. W. Badger, Introduction to Quantitative Aspects of Social Phenomena. New York , NY: Gordon and Breach, 1974.

[39] T. J. LaBonte, "Building a new performance vision: Break down organizational silos and create a unified approach to human performance improvement," Alexandria, VA: American Society for Training and Development, 2001.

[40] M. A. Diamond, H. F. Stein, and S. Allcore, "Organizational silos: Horizontal organizational fragmentation," Journal for the Psychoanalysis of Culture and Society, vol. 7, no. 2, pp. 280-296, 2002.

[41] R. K. Dell, "Current Issues—Breaking organizational silos: Removing barriers to exceptional performance," Journal-American Waterworks Association, vol. 97, no. 6, pp. 34-37, 2005.

[42] M. Schroeder, Fractals, Chaos, Power Laws. New York, NY: Freeman, 1991.

[43] R. D. Stacey, Strategic management and organisational dynamics: the challenge of complexity, 6th ed. Harlow, England: Pearson, 2011.

[44] M. Batty, P. A. Longley, Fractal Cities. San Diego, CA: Academic Press, 2004.

[45] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law," Contemporary Physics, vol. 46, no. 5, pp. 323-351, 2005.

[46] P. Andriani and B. McKelvey, "Beyond Gaussian Averages: Redirecting Organization Science Toward Extreme Events and Power Laws," Journal of International Business Studies, vol. 38, no. 7, 1212–1230, 2007.

[47] P. Andriani and B. McKelvey, "From Gaussian to Paretian Thinking: Causes and Implications of Power Laws in Organizations," Organization Science, vol. 20, no. 6, pp. 1053-1071, 2009.

[48] B. McKelvey and M. P. Salmador Sanchez, "Explaining the 2007 bank liquidity crisis: Lessons from complexity science and econophysics," Working paper, UCLA, Los Angeles, CA, 2011.

[49] B. McKelvey, B. B. Lichtenstein, and P. Andriani, "When organizations and ecosystems interact: Toward a law of requisite fractality in firms," International Journal of Complexity In Leadership and Management, vol. 2, no. 1-2, pp. 104-136, 2012.

[50] M. H. R. Stanley, L. A. N. Amaral, S. V. Buldyrev, S. Havlin, H. Leschhorn, P. Maass, M. A. Salinger, and H. E. Stanley, "Scaling Behavior in the Growth of Companies", Nature, vol. 379 no. 6568, pp. 804–806, 1966.

[51] R. V. Solé, D. Alonso, J. Bascompte, S. C. Manrubia, On the fractal nature of ecological and macroevolutionary dynamics, Fractals, vol. 9, no. 1, pp. 1-16, 2001.

 ISBN: 978-1-61208-341-4

[52] I. Ajzen, “The theory of planned behaviour: reactions and reflections,” Psychology and Health, vol. 26, no. 9, pp. 1113–1127, September 2011,

[53] L. Festinger,. A theory of cognitive dissonance. Stanford, CA: Stanford University Press, 1957.

[54] J. Holland, The global economy as an adaptive process, in The Economy as an Evolving Complex System, P. W. Anderson, K. J. Arrow, and D. Pines, (eds.), pp. 117-123. Redwood City, CA: Addison-Wesley, 1988.

[55] M. Uhl-Bien, R. Marion, and B. McKelvey, “Complex leadership: Shifting leadership from the industrial age to the knowledge era,” The Leadership Quarterly, vol. 18, no. 4, pp. 298-318, 2007.

[56] O. E. Williamson, Markets and Hierarchies. New York, NY: Free Press, 1975.

[57] D. W. Read, The utility of mathematical constructs in building archaeological theory. In: Voorrips, A. (Ed.), Mathematics and Information Science in Archaeology: A Flexible Framework, vol. 3, pp. 29-60. Bonn, Germany: Helos, 1990.

[58] J. B. Barney, “Firm resources and sustained competitive advantage.”, Journal of Management, vol. 17, no. 1, pp. 99-120, 1991.

[59] J. Barney, M. Wright, , and D. J. Ketchen, “The resource-based view of the firm: Ten years after 1991,” Journal of Management, vol. 27, no. 6, pp. 625-641, 2001.

[60] K. Eisenhardt and J. Martin, “Dynamic capabilities: What are they?,” Strategic Management Journal, vol. 21, no. 10-11, pp. 1105-1120, 2000.

[61] N. N. Taleb, The Black Swan: The Impact of the Highly Improbable Fragility. Random House, New York, 2007.

[62] Dooley, K. J., A. H. Van de Ven, “Explaining complex organizational dynamics,” Organization Science vol. 10, no. 3, pp. 358-372, 1999.

[63] D. Sornette, Why Stock Markets Crash. Princeton, NJ: Princeton University Press, 2003.

[64] De Vany, A., Hollywood Economics. Routledge: New York, 2004.

[65] D. Sornette, F. Deschâtres, T. Gilbert, and Y. Ageon, Endogenous versus exogenous shocks in complex networks, Physical Review Letters, vol. 93, no. 1, pp. 228701-228704, 2004.

[66] J. A. C. Baum and B. McKelvey, “Analysis of extremes in management studies.” In: D. J. Ketchen and D. D. Bergh, (Eds.), Research Methodology in Strategy and Management, vol. 3, pp. 123-197. Oxford, UK: Elsevier, 2006.

[67] M. J. Melitz, “The impact of trade on intra-industry reallocations and aggregate industry productivity,” Econometrica, vol. 71, no. 6, pp. 1695-1725, doi:10.1111/1468-0262.00467, 2003

[68] C. Anderson, The Long Tail. London, UK: Random House Business Books, 2006.

[69] B. McKelvey and P. Andriani, “Avoiding extreme risk before it occurs: A complexity science approach to incubation,” Risk Management, vol. 12, no. 1, pp. 54-82, 2010.

[70] P. Andriani and B. McKelvey, “Managing in a Pareto world calls for new thinking,” M@n@gement, vol. 14, no. 2, pp. 89-118, 2011.

[71] B. McKelvey, “Complexity leadership: The secret of Jack Welch’s success.” International Journal of Complexity in Leadership and Management, vol. 1, no. 1, pp. 4-36, 2010.

[72] A. J. P. Canuto, A. M. Campos, J. C. Alchiere, E. C. de Moura, A. M. Santos, E. B. dos Santos, and R. G. Soares, A personality-based model of agents for representing individuals in working organizations, in: Intelligent Agent Technology, IEEE/WIC/ACM International Conference, pp. 65-71, 2005.

[73] C. Bruun, "Agent-Based Computational Economics - An Introduction," working paper: Department of Economics, Politics and Public Administration, Aalborg University Aalborg, Denmark, 2006.

[74] P. Checkland, “Soft systems methodology: A thirty year retrospective,” Systems Research and Behavioral Science, vol. 17, pp. S11-S58, 2000.

[75] K. Lewin, “Action research and minority problems,” J. Soc. Issues, vol. 2, no. 4, pp. 34-46, 1946.

[76] D. Coghlan, “Action research: exploring perspectives on a philosophy of practical knowing,” The Academy of Management Annals, vol. 5, no. 1, pp. 53-87, 2011.

[77] C. Darwin, On the origin of species. London, UK: John Murray, 1859. Available from http://en.wikisource.org/wiki/On_the_Origin_of_Species_(1859)

[78] J. M. Baldwin, “A new factor in evolution,” American Naturalist, vol. 30, no. 354, pp. 441-451, 536-553, 1896.

[79] G. G. Simpson, The Baldwin effect, Evolution, vol. 7, no. 2, pp. 110-117, 1953.

[80] E. Crispo, “The Baldwin effect and genetic assimilation: revisiting two mechanisms of evolutionary change meditated by phenotypic plasticity,” Evolution, vol. 61, no. 11, pp. 2469-2479, 2007.

[81] A. V. Badyaev, “Evolutionary significance of phenotypic accommodation in novel environments: An empirical test of the Baldwin effect.” Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 364, no. 1520, pp. 1125-1141, 2009.

[82] S. A. Kauffman, “Foreword: Evolution beyond Newton, Darwin, and entailing law,” in B. G. Henning and A. D. Scarfe (Eds.), Beyond Mechanism: Putting Live Back into Biology, pp. 1-24. New York, NY: Lexington Books, 2013.

[83] A. C. Scarfe, “On the ramifications of the theory of organic selection for environmental and evolutionary ethics,” in B. G. Henning and A. D. Scarfe (Eds.), Beyond Mechanism: Putting Life Back into Biology, pp. 259-284. New York, NY: Lexington Books, 2013.

 ISBN: 978-1-61208-341-4

Towards Systematic Design of Adaptive Fault Tolerant Systems

Elena Troubitsyna, Kashif Javed
Åbo Akademi University, Finland
e-mails: {Elena.Troubitsyna, Kashif.Javed}@abo.fi

***Abstract*—The development of modern distributed software systems poses a significant engineering challenge. The system architecture should exhibit plasticity and high degree of reconfigurability to enable an automated adaptation to continuously changing operating conditions and component failures. Traditional engineering approaches are inefficient to cope with complexity of such systems to ensure their robustness and fault tolerance. Therefore, there is a clear need for the approaches explicitly addressing the problem of designing adaptive fault tolerance mechanisms. In this paper, we propose a systematic approach to the development of adaptive fault tolerant systems. We discuss the main principles of architecting such systems to enable plasticity and reconfigurability. We demonstrate how deployment of the predictive adaptation allows us to ensure that the system would be able to continuously deliver its services with the acceptable quality despite occurrence of component failures.**

Keywords-adaptable systems; fault tolerance, predictive adaptation; reconfiguration.

I. INTRODUCTION

The complexity of modern large-scale systems requires solutions that ensure that systems autonomously adapt to the operating environment and internal conditions. Often, such systems are put into a wide class of autonomic systems -- the software-intensive systems that, besides providing their intended functionality, are also capable to diagnose and recover from errors caused either by external faults or unforeseen state of environment in which the system is operating [3]. In this paper, we focus on the fault tolerance aspect of such systems.

Fault tolerance is an ability of a system to deliver its services in a predictable way despite faults [8]. The generic principle underlying design of fault tolerant systems is to detect a discrepancy between a model representing fault free system behaviour and the observed state, and implement error recovery [8] .

In this paper, we propose a general pattern for architecting and developing the adaptive fault tolerant systems. The proposed pattern supports a layered design approach [6] that enables separation of concerns and facilitates structured design of fault tolerance mechanisms. In our representation of the architectural pattern, we define the interfaces between the components at different levels of abstraction to ensure correct propagation of fault tolerance related data. The high-level coordination of the fault tolerance mechanisms is implemented by an adaptation manager – a component that is responsible for implementing predictive fault tolerance. To specify the adaptation manager, we propose an algorithm that allows the adaptation manager to monitor state of the system at the run time and implement proactive adaptation. Such an approach ensures that the overall system would continuously deliver the services with the acceptable quality. We believe that the proposed approach ensures a systematic development of adaptive fault tolerant systems.

The paper is structured as follows: in Section II, we overview the state-of-the-art in designing adaptive fault tolerant systems. In Section III, we describe general principles of achieving fault tolerance, and, in particular, proactive fault tolerance. In Section IV, we present our proposal for structuring adaptive fault tolerant system. In Section V, we present our proposal for algorithms that implement proactive fault tolerance. Finally, in Section VI, we discuss the proposed approach and future work.

II. RELATED WORK

The need for high performance and continuous service provisioning demands novel solutions for achieving system fault tolerance. We are increasingly observing deployment of proactive fault tolerance techniques that replace traditional reactive approaches [10]. In modern large-scale systems, error rate is increasing and reliance on traditional "error-detection – error-recovery" pattern leads to poor performance and prolonged system downtime, which is often unacceptable. The approaches for proactive fault tolerance are based on preventive treatment of faults aiming at precluding failures and minimising recovery time [10]. The main mechanism of achieving proactive fault tolerance is adaptation.

The problem of software adaptation has been extensively studied at the implementation level, (see e.g., [2] for an overview). However, there is a lack of approaches that attempt to derive appropriate adaptation mechanisms from system-level goals as well as support layered reasoning needed to efficiently cope with system complexity. A prominent work on formal modelling of adaptive systems has been done within the HATS project [2]. In [13][14], an approach to quantitative assessment of reconfiguration strategy has been proposed. In our previous work, we also investigated the impact of faults on dependability, as well as

 ISBN: 978-1-61208-341-4

structured approach to designing fault tolerant distributed systems [7][11].

Current engineering practice takes an architecture-centric perspective on adaptive systems. Among the most prominent examples are the Rainbow framework proposed at Carnegie Mellon University [12] and the autonomic computing initiative by IBM [3]. These frameworks outline the main abstractions for describing and managing dynamic system changes. However, currently, the approaches to proactive fault tolerance are not well-integrated into the system development process [10]. In this paper, we will address this problem by proposing a structured approach to architecting adaptive fault tolerant systems. Our approach aims at facilitating design space exploration at the early development stages and enabling explicit representation of the mechanisms for proactive fault tolerance.

III. FAULT TOLERANCE

The main goal of introducing fault tolerance is to design a system in such a way that faults of components do not result in a system failure. A fault cannot be detected by a system until the manifestation of the fault generates *errors* in the component function. The first step in implementing fault tolerance is error processing [10]. *Error processing* aims at removing errors from the computational state.

The first step in error processing is *error detection*. An error is a manifestation of a fault. The general mechanism of error detection is to intercept outputs produced by a system (or a component) and to check whether those outputs conform to the specification of fault free behaviour. Discrepancy between produced outputs and the specification indicates an occurrence of an error. The next step in error processing – *damage confinement* – is concerned with structuring the system to minimise the spread of errors. Once the damage is assessed and confined the error recovery can be performed. *Error recovery* has two main forms – forward and backward error recovery. The forward error recovery mechanisms manipulate the current system state to produce a new system state, which is presumably error free. The success of error recovery strongly depends on how precisely the error is located and how well it is confined. A typical example of forward recovery is *failsafe* [1]. If a system has a safe though non-operational state then it may be possible to recover from an error by forcing the system permanently to that safe state (obviously, this strategy is only appropriate where shut down of the system operation is possible).

By analyzing actions to be undertaken for error processing, we observe that error processing imposes additional requirements on the system design. Namely:

- The system should be specified in such a way that error occurrence conditions are easily deduced and then explicitly checked;
- The system architecture should enable error confinement;
- Error recovery procedures should be identified for every output, which differs from the specified one.

Obviously, an incorporation of error processing in the system design has a strong impact on all levels of the system structure. Hence, fault tolerance should be an intrinsic part of system development and should start from the early stages of the system design.

To embrace complexity challenge, fault tolerance community has been proposing new concepts that can be seen from initiatives and research efforts on autonomic computing [3] and various forums on self-healing [9] or self-protection (see, e.g., [1]). These terms span a wide range of research fields ranging from adaptive memory management to advanced security mechanisms.

A promising direction among them focuses on determining how computer systems can proactively handle failures: if the system knows about a critical situation in advance, it can try to apply countermeasures in order to prevent the occurrence of a failure, or it can prepare repair mechanisms for the upcoming failure, in order to reduce the time-to-repair.

Such an approach can be called proactive fault tolerance. It encompasses three main steps:

1. Failure prediction: it aims at identifying failure-prone situations, i.e., the situations that will probably evolve into a failure. The result of failure prediction is an evaluation of whether the current situation is failure-prone.

2. Proactive reconfiguration: based on the outcome of failure prediction, a system should make a decision and implement the countermeasures to be executed in order to remedy the problem. These decisions are based on an objective function taking into account the cost of the actions, the confidence in the prediction, and the effectiveness and complexity of the actions to determine the optimal tradeoff. Challenges for action execution include online reconfiguration of globally distributed systems, data synchronization of distributed data centers, and many more.

3. Recovery: this stage enables graceful degradation of services while the resources are insufficient for mitigating the failures. For instance, the predictive reconfiguration might not be completed as promptly as expected and the system should compensate for insufficient resources. Another example would be a sudden simultaneous failure of several components due to unexpectedly adverse situations in the environment.

Each one of these stages is important for an efficient implementation of the proactive fault tolerance. Hence, novel architectural solutions, algorithms and development approaches are needed to attain the goal of building adaptive fault tolerant systems.

 ISBN: 978-1-61208-341-4

To build a proactive fault tolerance solution that is able to boost system dependability, the best techniques from all fields for the given surrounding conditions have to be combined.

In this paper, we consider the proactive fault tolerance to be the main adaptation mechanism to achieve system dependability. In the next section, we present our approach to structuring an adaptive fault tolerant system. Then, we focus on designing the proactive adaptation mechanisms. Our proposal aims at enhancing self-adaptation system capabilities. Our goal is to design the mechanisms that allow a system to autonomously adapt to changing operating conditions without human intervention. Essentially, our proposal follows a spirit of the autonomic computing paradigm.

IV. ARCHITECTURE OF ADAPTIVE FAULT TOLERANT SYSTEMS

In this paper, we propose to structure an adaptive fault tolerant system in a layered manner [6]. The layered architecture significantly simplifies the development of complex software-intensive systems. Each layer becomes responsible for a certain aspect of the system behaviour. It facilitates a clear separation of concerns and simplifies the interfaces between the layers. The main issue is to device a well-structured clean architecture that does not introduce tangled interdependencies between layers. In this paper, we propose to structure the architecture of a fault tolerant adaptive system in four layers:

- Application layer
- Adaptation layer
- Fault tolerance layer
- Physical layer

The physical layer represents the environment whose state should be monitored. It might be a complex control system that uses sensors to monitor the health of its components. Another example might be an indoor sensor network that monitors such conditions as temperature, humidity, the level of CO, etc. Finally, it might also be a sensor network for monitoring the outdoor environment, e.g., such as used for forest fire detection, air pollution etc.

The fault tolerance layer performs the data aggregation and evaluation of the quality of monitoring. This information is supplied to the adaptation layer that is responsible for defining the proactive adaptation policy. The aim of the application and fault tolerance layer is to continuously supply the application with the monitoring data of an acceptable quality. The design of the application is defined by its purpose – it varies from the complex control functions to collecting data intelligence. The graphical representation of the system architecture is given in Fig.1.

The physical layer consists of the component to be controlled by the application software. In order to implement proactive fault tolerance, the software should continuously monitor the state of the controlled components.

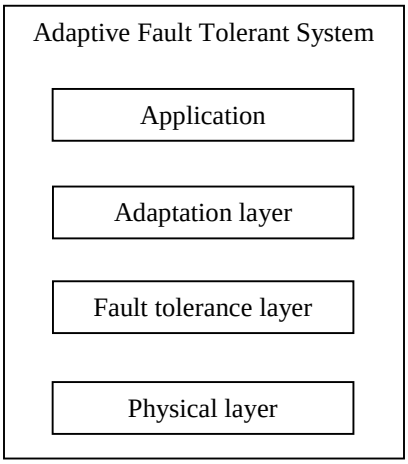


Figure 1. Structure of an adaptive fault tolerant system.

The monitoring capabilities are achieved by integrating sensors that measure the parameters required to observe the behaviour of the system in real-time. Usually, complex systems contain a large number of sensors. Hence, from the fault tolerance perspective, the physical layer can be considered as a sensor network.

It generates raw data. Each sensor produces the data in the following format

<value, timestamp>

We consider two most typical failure modes of the sensors: *stuck at previous value* and producing a (detectably) *incorrect value*. In the former case, the sensor fails silently by failing to update its reading, i.e., the timestamp indicates that the produced data is old. In the latter case, the sensor produces the value that is outside of the feasible range.

At the fault tolerance layer resides fault tolerance manager. The goal of the fault tolerance manager is

- To periodically read the sensor data,
- To filter out faulty data,
- To compute the average value of valid data together with defining the quality level.

The fault tolerance manager produces the input for the adaptation manager as a tuple

<value, level>

To compute the quality level, the fault tolerance manager keeps track of the number of sensors that have produced valid data. There are two thresholds: *lim1* and *lim2* such that *lim2* > *lim1*. They determine the quality level. If the number of the sensors that produced the valid data is greater than *lim2* then the quality level is set to *Level 3*. If the number of sensors produced valid data is between *lim1* and *lim2* then

 ISBN: 978-1-61208-341-4

the quality level is set to *Level 2*. If the number of valid readings is between *1* and *lim1* then the quality level is set to *Level 1*. Finally, if none of the sensors have produced valid results then the quality level is assigned value *Level 0*.

The adaptation manager and deployment manager constitute the adaptation layer. The adaptation manager receives the data from the fault tolerance manager in the format

<value, level>

where *level* is an integer between *0* and *3*. If the level has value *3*, then, the value has a good quality and the adaptation manager simply forwards the received value to the applications. However, if the quality level is below *3* but greater that *0* then the adaptation manager still forwards the received data to the application but starts an observation period.

The aim of the observation period is to establish whether the decline in the quality of data is temporal or permanent. Assume that, after receiving a value with the levels *1* or *2*, the adaptation manager observes a continuous period of receiving data with quality level *3*. Then, the observation period terminates and no reconfiguration is initiated, i.e., the adaptation manager treats the decline in the quality of data as a temporal one and considers the system to be healthy.

If, during the observation period the adaptation manager continuously receives data with quality level *1* or *2* then after the observation period expires, it initiates reconfiguration, i.e., considers the quality deterioration to be the permanent one.

The reconfiguration is triggered by sending a request to the *deployment manager* to deploy a new set of sensors. The deployment can be achieved in several different ways. For instance, if we consider a wireless sensor network that is used to monitor the state of the environment then the deployment is performed via a distribution of a set of fresh sensors (e.g., from an airplane). If the sensors are used to monitor an indoor environment then the deployment triggers a request to the maintenance company. The same principle applies if the sensor network is used to monitor the behaviour of a complex control system. In any case, the main advantage of the proposed approach is a possibility to preventively react on the deterioration of the quality of monitoring and avoid the loss of the observability of the physical layer.

The requested number of new sensors to be deployed depends on how deeply the level of data quality has deteriorated. If the quality level has value *1* then the deployment manager requests *n* new sensors to be deployed. If the quality level has the value *2* then *m* new sensors are to be deployed, where $m<n$.

In general, we could design a more sophisticated deployment mechanism. For instance, if each sensor or a group of sensors is assigned an id then the failures can be diagnosed precisely. This would allow the adaptation manager to communicate the exact requirements for the deployment of new sensors.

When the new sensors are deployed, the deployment manager acknowledges the completion of the reconfiguration and the adaptation manager notifies the fault tolerance manager about availability of the new sensors. The fault tolerance manager closes the connection with the failed sensors and establishes connection with the newly deployed ones.

An important aspect to be considered is how to define the behaviour of the adaptation manager when the quality level keeps fluctuating between the values 2 and 3. On the one hand, the adaptation manager should not trigger the reconfiguration prematurely. On the other hand, delaying a reaction on such an unstable situation might result in an abrupt deterioration of the quality of data that should be prevented.

To resolve this issue, we let the adaptation manager to maintain the observation period as long as no continuous improvement in quality has been observed. Every time when the data are received with the quality threshold lower than *3*, the adaptation manager increments the counter of the observation period. When this counter exceeds the predefined threshold, the adaptation manager triggers the reconfiguration. This approach is taken to ensure that the preventive reconfiguration will be initiated even if the system keeps fluctuating between quality levels.

Finally, if the adaptation manager receives data with the quality level equal to *0*, then it immediately initiates reconfiguration of the data flow. In this case, it starts to send to the application data received at the previous cycle. It continues to send the last data with an acceptable quality value until the reconfiguration is completed and the fault tolerance manager starts to send the data with an acceptable quality level.

In the next section, we define the main behavioural patterns of adaptation manager and fault tolerance manager.

V. ALGORITMS FOR PROACTIVE FAULT TOLERANCE

Let us focus first on defining the module specifying the fault tolerance manager.

The module should implement the procedures of

- Reading the sensor data,
- Checking validity of sensor data with respect to time and feasibility
- Calculating the average of the received valid data and the quality level.

In our definition of the fault tolerance manager, we used two abstract functions *fresh* and *valid*. The function *fresh* relies on the specific parameters to determine whether the produced data is fresh. Since the clocks of the sensors might fluctuate, the function checks whether the timestamp is within certain boundaries.

The function *valid* checks feasibility of the data produced by a sensor. It returns the Boolean value *True* if the data is valid and *False* otherwise.

 ISBN: 978-1-61208-341-4

```
Module Fault Tolerance Manager
Global Variables
   in_buffers: array of  <float, INT>
   out_buffer: seq of <float, INT>

Local Variables
   count: INT /*counter of healthy sensors
   sum : float /*sum of readings
   avg: float /*average value
   level:  [0..3]

Initialisation:
   count:= 0;
   sum:= 0;
   avg:= 0;
   level:= 0

Begin

  for i = 1 to k do
    read (data, time_stamp, in_buffer[i]);
    if
        fresh (time_stamp) = True & valid(data)= True
    then  count:= count +1; sum := sum +data
    end;

 if counter > 0 then avg:= sum/count;

 case count = 0 then level:= 0
   elseif count>0 & count<lim1 then level:=1
   elseif count>lim1 & count<lim2 then level:=2
   else level:=3;

 out_buf:= out_buf^<avg,level>;
 count:= 0;
 sum:= 0;
 avg:= 0

End
```

Figure 2. Fault Tolerance Manager.

Reliance of the abstract functions allows us to parameterise the definition of the module and reuse the proposed definition in different contexts.

In our definition of the module, we have abstracted away from the implementation details of the communication between the fault tolerance manager and the sensors. We assume that they communicate by shared variables -- data and time stamps that are stored in the *in_buf* array of pairs.

The proposed algorithm implements the procedure of reading the sensor data, checking their validity with respect to time and feasibility and calculates the average of the received valid data.

By keeping track of the number of valid readings, the fault tolerance manager calculates the quality level. It compares this number with two constants – *lim1* and *lim2*. The pair of calculated data and the quality level is appended to the output buffer that is read by the Adaptation Manager. The specification of the Fault Tolerance Manager module is given in Fig. 2 and the Adaptation Manager in Fig. 3.

```
Module Adaptation Manager

Global Variables
    a_out_buf: float

Local variables:
    observ : Bool
    cur_level : INT
    cur_data:float
    fault_count : INT
    suc_count : INT
    mode: {Normal, Adapt, Adapt_Compl, Adapt_activ}

Initialisation:
    observ :=0;
    cur_level :=0;
    fault_count :=0;
    suc_count :=0;

Begin

 cur_level, cur_data := head(out_buf);

if observ= False & cur_level= 3 then out_buf:= cur_data

if observ= False & cur_level= 2 & fault_count<thr
 then fault_count:= fault_count+1; out_buf:= cur_data;

if observ= False & cur_level<3 & cur_level>0 &
   fault_count>thr-1
 then mode := adapt_active, adapt_req:= True;

if observ= False & cur_level=3 & fault_count>0 &
    fault_count<thr-1
then observ:= True; suc_count := suc_count +1;
    observ:= 0;

if observ= True & cur_level=3 & fault_count>0 &
  fault_count<thr-1 & suc_count<thr_s
then suc_count:= suc_count+1;
    observ_s_iter:= observ_s_iter:=+1;

if observ= True & cur_level=3 & fault_count>0 &
  fault_count<thr-1 & suc_count>thr_s-1 &
  suc_count =observ_s_iter
then observ:= False ; suc_count:= 0; fault_count:= 0;
      observ_s_iter:= 0;

if observ= True & cur_level<3 & fault_count>0 &
   fault_count<thr-1 & suc_count<thr_s
then suc_count:= suc_count+1;
    observ_s_iter:= observ_s_iter+1;

if mode= adapt_activ then adapt_req ;

if adapt_conf  then mode:= normal
End
```

Figure 3. Adaptation Manager.

In the specification of the Adaptation manager, the variable *observ* indicates whether the observation period has started. The variable obtains the value *True* when the first

data with the quality level below *3* is received. The variable is reset to *True* if the quality has recovered or a new period of observation is initiated.

The variables *cur_level* and *cur_data* designate the data and the quality level received from the fault tolerance manager. The variable *fault_count* is used to keep track of the number of iterations, in which the data with the quality level lower than 3 have been received. When the value of *fault_count* exceeds the predefined threshold *thr*, the reconfiguration is triggered.

The variable *suc_count* is used to keep track of the iterations that produced data with the quality level 3 after the observation period has been initiated. When the value of *suc_count* exceeds the predefined threshold *thr_s* the adaptation manager has continuously received the data with the quality level 3 for sufficiently long period of time. Therefore, the quality level has recovered and the observation period can be deactivated.

The adaptation manager provides the application with the latest data by updating the global variable *a_out_buf*. It forwards the data received from the fault tolerance manager if the quality level is higher than zero. Otherwise, it simply does not update the variable.

The adaptation manager triggers the reconfiguration by issuing the adaptation request *adapt_req* that is received by the deployment manager. When the new sensors are deployed the deployment manager confirms the reconfiguration by issuing the signal *adapt_conf*.

After triggering the reconfiguration, the adaptation manager enters the mode *Adapt*. After the reconfiguration is completed, the adaptation manager enters the mode *Adapt_Compl*. In this mode [4] [5], it notifies the fault tolerance manager about availability of new healthy sensors. As a response to this, the fault tolerance manager shuts down the connection with the failed sensors and establishes a new connection with the newly deployed sensors. After this procedure is completed, the fault tolerance manager notifies the adaptation manager. It enables transition to the mode *Normal*.

The general scheme of an implementation of the mode transition is given in Fig. 4. The main principle that underlies the mode transition is as follows: the mode is stable and unchanged until a fluctuation in the quality level is registered. We show the snippet implementing this principle as a generic mode changing procedure.

The proposed architecture ensures a separation of concerns and clear allocation of responsibilities between the components. Indeed, the fault tolerance manager is responsible for collecting data and validating them. It encapsulates the failures of sensors and gives only the high-level indication of the current health of the system by annotating the data with the quality level. The adaptation manager is responsible for diagnosing the situation and executing the preventive reconfiguration – requesting the new sensors to be deployed before the quality of data deteriorates below the acceptable level. At the same time, it also ensures remedial actions when no data is produced – it outputs to the application the last healthy value. Such behaviour ensures graceful degradation of quality of service.

Procedure *ModeTransition*

```
Variables
  last_mode: {Normal, Adapt, Adapt_Compl, Adapt_activ}
  next_target: {Normal, Adapt, Adapt_Compl, Adapt_activ}
  prev_target: {Normal, Adapt, Adapt_Compl, Adapt_activ}
  level: int

Begin

if adaptation completed
then initiate a forward transition
       to next_target according to
       the predefined scenario;

if level dropped
then initiate a backward transition to next_target
       adaptation mode
     The choice of target mode depends on severity
     of level decrease;

 if the conditions for entering the target
   mode are satisfied
then complete a transition to next_target mode
       and become stable ;

if neither the conditions for entering
 the next global mode are satisfied nor the level dropped
then maintain the current mode

End
```

Figure 4. Mode transition procedure.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a systematic approach to architecting adaptive fault tolerant systems. We have demonstrated how to structure the system to facilitate layered design of proactive fault tolerant mechanisms. We defined the information flow between the layers of the system architecture that enables adaptation and guarantees a continuous delivery of services with an acceptable quality level.

Proactive fault tolerance is a promising research direction that aims at providing systems with capabilities of executing preventive reconfiguration to preclude occurrence of failure and disruption in service provision. In our paper, the main mechanism of achieving proactive fault tolerance relies on several levels of error detection and monitoring of system health.

As a future work, we are planning to investigate alternative approaches to preventive reconfiguration as well as conduct quantitative assessment of various system characteristics, e.g., correlation between frequency of the network rejuvenation with new sensors and quality of data, proportion between periods of low quality data and different thresholds etc. Such a work, would allow us to define heuristics for designing proactive fault tolerance.

 ISBN: 978-1-61208-341-4

REFERENCES

[1] O. Babaouglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen (Eds.) *Self-Star Properties in Complex Information Systems*. LNCS 3460. Springer-Verlag, 2005.

[2] *HATS Project*: Highly Adaptable and Trustworthy Software using formal models. *www.hats-project.eu/*.Accessed 20.03.2014

[3] P. Horn, Autonomic Computing*:* IBM's perspective on the State of Information Technology. http://researchweb.watson.ibm.com /autonomic/. Accessed 20.03.2014

[4] A .Iliasov, E. Troubitsyna, L. Laibinis, A.Romanovsky, and K.Varpaaniemi. Verfifying Mode Consistency for On-Board Satellite Softyware. In.Proc. SAFECOMP 2010, LNCS 6351, pp. 126-141, Springer, 2004.

[5] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K.Varpaaniemi, D. Ilic, T. Latvala, Developing Mode-Rich Satellite Software by Refinement in Event B . In: *Proc. of FMICS 2010*, LNCS 6371, pp. 50-66, Springer, 2010.

[6] L. Laibinis and E.Troubitsyna. Fault tolerance in a layered architecture: a general specification pattern in B. In Proc. of SEFM 2004. pp. 346-355, IEEE Computer Press, 2004.

[7] L. Laibinis, E. Troubitsyna, A. Iliasov and A. Romanovsky. Rigorous Development of Fault-Tolerant Agent Systems. *Rigorous Development of Complex Fault-Tolerant Systems*. LNCS 4157, pp. 241-260, Springer, 2006.

[8] J. C. Laprie, *Dependability: Basic Concepts and Terminology*. New York, Springer-Verlag, 1991.

[9] M. Salehie, L. Tahvildari: Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 4(2). ACM, 2009.

[10] F. Salfner, M. Lenk, and M. Malek: A survey of online failure prediction methods. *ACM Comput. Surv*. 42(3), 2010.

[11] K. Sere and E. Troubitsyna. Safety Analysis in Formal Specification. In *Proc. of FM'99*, LNCS 1709, pp. 1564 – 1583, Springer, 1999.

[12] B. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan. Discovering Architectures from Running Systems. In *IEEE Transactions on Software Engineering*, Vol. 32(7), July 2006.

[13] A. Tarasyuk, I. Pereverzeva, E. Troubitsyna, T. Latvala, and L. Nummila, Formal Development and Assessment of a Reconfigurable On-Board Satellite System. In Proc. of *SAFECOMP 2012*, LNCS 7612, pp. 210–222, Springer-Verlag, 2012.

[14] E. Troubitsyna. Reliability assessment through probabilistic refinement. Nordic Journal of Computing 6(3), 320-342, 1999.

 ISBN: 978-1-61208-341-4

Moving Towards a Distributed Network of Proactive, Self-Adaptive and Context-Aware Systems

Remus-Alexandru Dobrican, Denis Zampunieris
Computer Science and Communication Research Unit, University of Luxembourg
Luxembourg, Luxembourg
Email:{remus.dobrican, denis.zampunieris}@uni.lu

***Abstract*—Instead of being static and waiting passively for instructions, software systems are required to take a more proactive approach in their behavior in order to anticipate and to adapt to the needs of their users. To design and develop such systems in an affordable, predictable and timely manner is a great engineering challenge. Even though there have been notable steps towards distributed self-adaptive and context-aware systems, there is still a lack of methodologies on how to model and implement applications which have to distribute and to manage large amounts of information. In this work-in-progress, we address this issue by proposing a self-adaptive and context-aware model with a structure that allows the system to learn from the user's behavior by using Proactive Computing. The novelty comes from the possibility of having a distributed network of Proactive Engines in which the exchange of contextual information would help each system to take smart decisions.**

Keywords—self-adaptive systems; context-aware systems; proactive computing; distributed network.

I. INTRODUCTION

The demand for devices and applications that are able to adapt their behavior at run-time, as a response to the increasing demands of users, has risen considerably in the last couple of years [1]. Giving instructions to complex software systems is becoming quite a difficult task for users, as it requires their continuous involvement, a set of advanced technical skills and a lot of knowledge about the system. As a consequence, our model is leading the users towards new ways of interacting with smart systems that will be able to perform a variety of automated tasks on users' behalf.

Three main properties are to be distinguished when speaking about systems that dynamically adapt themselves according to the context variation or the requirements change: self-adaptation, proactivity and context-awareness.

Self-adaptation in software systems comes in many different aspects. Self-adaptive systems can be characterized by their operating mode which easily permits them to fulfill their goals in a modified context. Feedback loops provide an architectural solution for self-adaptation. Brun et al. [2] indicate that feedback loops usually include four key activities: *collecting*, *analyzing*, *deciding* and *acting*. These activities are essential for achieving self-adaptability. In Figure 1, a generic model of a unidirectional feedback loop is given. It shows the inputs or the outputs of each state but the data flow between the states is omitted.

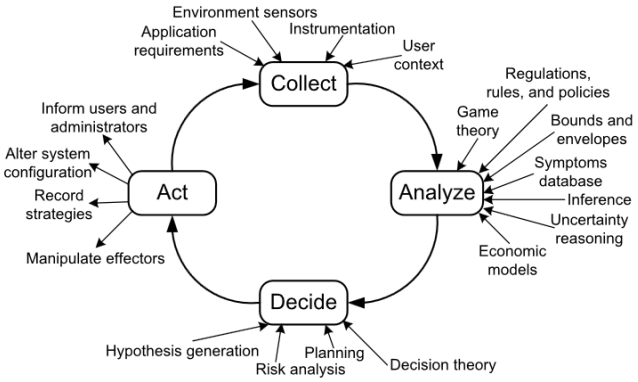


Figure 1. Autonomic control loop [3]

Context-aware systems are designed to continuously analyzing contextual information, which is a key feature for determining the occurrence or the lack of events.

Events play a central role in the lifecycle of software systems. They range from simple request for different services to serious incidents that prevent the well-functioning of a system. Events can be divided into three main categories: foreseen (*taken care of*), expected (*planned for*) and unexpected (*not planned for*) [4].

Tennenhouse [5] firstly introduced Proactive Computing as a new mode of operation that was crucial for moving towards human-supervised computing. The essential features of proactive systems, as seen in [6], are taking decision for their users and acting on their own initiative. Proactive Computing is a solution for foreseeable events, while context-awareness and self-adaptiveness handle unforeseen events, which are seen as deviations from the normal situations.

The contribution of this paper is two-fold. First, it offers an infrastructure for software systems capable of performing automated tasks for the user, of analyzing large quantities of data and making decision in different contexts. Second, it provides an analysis of a distributed network of systems that are implementing our model.

The rest of the paper is organized as follows: Section 2 describes the main characteristics of a Proactive Engine. Section 3 investigates the possibility of having a distributed network of Proactive Engines. Section 4 provides an example of application for our model; other applications are proposed. Section 5 conclusions about the potential of Proactive Engines.

 ISBN: 978-1-61208-341-4

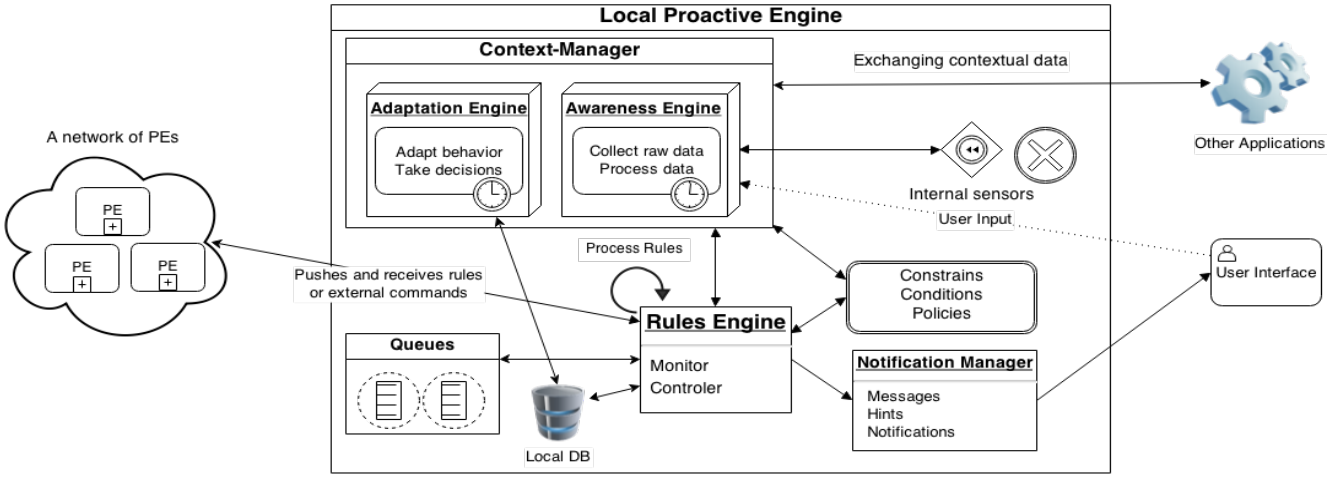


Figure 2. The infrastructure of a Proactive Engine

II. PREVIOUS WORK

Zampunieris developed the concept and the structure of the first Proactive Engine in 2006 [7]. It was designed as a complex mechanism for running Proactive Rules. A Proactive Rule is a structure conceived to perform specific actions in case a special situation was detected or in case of the lack of an event. The detection of students that did not submit their online assignment and the notification of their professor as a consequence, is a concrete example of a rule which was used in a real-case scenario, when the initial Proactive Engine was deployed aside a Learning Management System (LMS) [8]. Results showed that major limitations of a LMS such as the restricted interaction and limited collaboration between learners and educators inside courses could be overcome with the help of a Proactive Computing [9]. Previous work, [10] and [11], focused until now on applying Proactive Computing on a single system, thus exploring only the scenario of having only one centralized Proactive Engine. But, a centralized solution can become quite fast non-scalable in many scenarios where a Proactive Engine handles a big number of devices and applications. The possibility of having an entire network of Proactive Engines exchanging data and learning from each other was not yet explored.

III. PROACTIVE ENGINES

We propose a new version of the Proactive Engine, where processes are divided between the sub-parts of the model. Before, Proactive Rules were taking care of data acquisition, activation guards, conditions, actions and rules generation; now, each step is assigned to a specific structure. A major benefit of separating these processes is that they are handled by structures that are focusing only on particular tasks.

In order to develop a proactive context-aware adaptive system, an infrastructure that combines and uses all three properties is required. The LPE is an advanced mechanism that could be easily integrated into new software systems because it provides means for gathering data from the internal and external sensors, for detecting context changes, for processing and modeling contextual information, for executing adaptive tasks and for providing an adequate system behavior in any situation. The term "sensor" refers not only to the hardware parts being able to sense but also to the various data sources that may give contextual information.

Thus, the architecture of a LPE is composed of a set of interconnected components, including a Context-Manager, a Rules Engine connected to a set of Queues and a local database, and a Notification Manager (as seen in Figure 2).

A. The Context-Manager

The Context-Manager is mainly responsible for detecting and handling context changes that appear, and as a result, taking the proper actions. Another important task for the Context-Manager is to acquire user input and to decide if it is relevant or not. It is composed two elements: the Awareness Engine and the Adaptation Engine.

1) The Awareness Engine

This component is managing the data coming from sensors, which are in charge of detecting possible context changes. For smartphones, sensors are providing important information about the user's location, motion and preferences. For PCs, the information would focus more on the user's interests, activities and set of used applications. Accessing this kind of information should be limited to some extend and controlled as it represents a privacy issue.

2) The Adaptation Engine

This component is crucial, as it is used for dealing with *unexpected events* and for ensuring that adaptive actions are performed in a smooth cooperation between the main sub-parts of the Proactive Engine. Also, it has to check the constraints and the conditions of the system before adaptation and if the system will still behave according to its policies.

 ISBN: 978-1-61208-341-4

B. The Rules Engine

The Rules Engine is responsible for maintaining a precise overview of the system's goals and for running Proactive Rules. It keeps a list of required actions that would come as a response in case an *expected event* shows up. It is also used for storing the state of the system. Executing multiple Proactive Rules in parallel is due to its integrated Queue System and it is one of the great functionalities of the Rules Engine. Proactive Rules can be used for serving multiple purposes: for checking context situations, for detecting special events, for analyzing contextual information, for synchronizing sub-parts of the model, for saving useful data into the Local Database, for sending rules and commands to other PE and for sending content to the Notification Manager. The Awareness Engine and the Adaptation Engine have the ability to activate Proactive Rules.

C. The Notification Manager

The purpose of the Notification Manager is to deliver informative content to the user. The content can take various forms like hints, messages, notifications or alarm. This is a crucial part of the entire model as it helps in achieving his/her goals, guides him/her in multiple situations and informs the user about certain events.

IV. A NETWORK OF PROACTIVE ENGINES

PEs are designed to work both offline and online. Having a network of distributed LPEs that communicate and exchange data provides a great opportunity for these systems to gain useful information. This way, LPEs are not only gathering data from their internal sensors but also from other LPEs. By design, information sharing between devices using LPEs is conceived to be done in a transparent way, without the implicit command of the user.

Figure 3 shows a possible scenario of a network of distributed LPEs. Three devices, with a running LPE, located on the same LAN, are connected to the Internet through a WiFi connection. A direct connection can be also established via Bluetooth, via Near Field Communication (NFC) techniques or via Android's WiFiP2P library for smartphones. The advantage of having a direct connection between the devices, illustrated in figure 3 with a straight line, is that Proactive Rules are exchanged immediately, without having to be sent firstly to a server. This means that each device with a LPE will be acting like a server, being able to receive and send data to other devices with LPEs.

The most significant aspect to be taken into consideration is the actual information that is gained by a LPE when it gets data from other LPEs. One case is to find common interest or preferences between users that are working with applications having an integrated LPE. For example, a user could be looking for a ride on a car-sharing web site. Another user, which would be located nearby, maybe from the same city, would be looking for a ride having the same destination and exactly on the same dates. The LPEs would notify both users and would propose to share a ride for

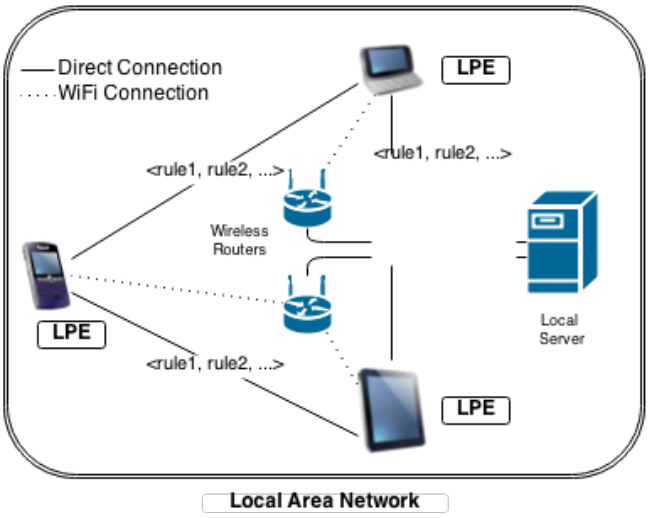


Figure 3. A possible network of distributed LPEs

reducing the costs. Another case where data exchanging is useful is when a LPE is not sure what action to take and how to adapt its behavior when *unexpected events* are appearing. Requesting feedback from other LPEs that have more information is a possible solution for taking the right decision.

If we take, for example, two LPEs, one which was offline for a long period of time and one which was online during the same period of time. And now, both of the LPEs would be able to share information because they would be have access to a communication channel between them. The LPE that was offline could learn a lot from the online LPE that stored information about its previous tasks and about the older state of the system, without using the Adaptive Engine, the Awareness Engine and the Rules Engine to process similar data and to go through the same adaptation process. As a consequence, local resources and time could be saved.

V. CASE STUDY

To better illustrate the behavior of a LPE and the usefulness of having a network of LPEs, we created an example of a possible scenario for its practical implementation. For simplicity, we focused more on describing the possible situations that highlight the benefits of having a network of LPEs and not on the implementation details.

All around the world, students are using online e-learning platforms, like Moodle™ [12], for accessing educational content, completing assignments and participating in discussion related to their courses. These e-learning platforms are quite static as they are waiting for instructions or commands from their users. This is why an e-learning application for PCs and for smartphones, with an integrated Proactive Engine, would come in hand. We assume that the application would be directly connected with the web platform and would have access to all the data from the student's account on the LMS.

 ISBN: 978-1-61208-341-4

The application would include basic actions like displaying notifications and questions for the user, provide hints and trigger alarms. Hints would be used for guiding the user, questions for asking for specific instructions, notifications as short messages to inform the user and alarm to alert him/her in case of extraordinary situations or/and events.

Even though these actions are quite elementary, they are already addressing some of the major issues when using an online e-learning platform. These issues appear because of the lack of an immediate notification channel between the students or between the students and the professors in case extraordinary situations appear. Certain online platform have an online mechanism for enrolling to an exam, and students often miss these deadlines, resulting in a big problem both for the student and the administration of universities and schools. More issues include missing deadlines for assignments and nonparticipating in forums.

For example, if an instructor were to give an exam on a specific date, at a specific hour, and is late due to traffic, he/she could post a short message, via his/her smartphone, on the forum of the course announcing that he/she will be late. Not only will the students be notified of this, but a person from the administration could also alert the students in person if they would not have their device with them. The sensors of the LPE would sense that he is moving and so would adjust the graphical user interface for writing messages.

More advance actions would include setting an alarm for deadlines, putting the events into an integrated calendar, proposing to students to collaborate on solving assignments with other classmates which are close to their location or even more, automatically download documents or course material directly to the private PCs or smartphones of the students. The majority of these actions are not currently provided by any existing LMS and, adding plugins or third party applications will not change the overall behavior of the system.

In Figure 4, an example of a Proactive Rule, which would be used for this case study, is illustrated in pseudo-code. More specifically, it is a Global Meta-Scenario because it runs at each iteration of the Proactive Engine and because it is used only when there are at least two LPEs on the same network. Its purpose is to invite the users of the LPEs, in case they are working on the same assignment, to collaborate and share their knowledge. The Proactive aspect comes from the fact that this situation is anticipated by the Global Meta-Scenario, without any specific intervention or command from the users of the LPEs.

There are five main parts that compose a Proactive Rule: data acquisition, activation guards, conditions, actions and rules generation. The first part is used for gathering useful data, in this case if there are new connections or LPEs available on the same network, the second and the third part are used for checking for special conditions and constraints, like if the users of the LPEs have common assignments, and the fourth and the fifth parts are used to take specific actions, like sending a personalized messages to the users of the LPEs, and to generate other Proactive Rules.

```
Global Meta-Scenario (GMS) 001
Description: This Rule is designed to run on each
LPE in order to check for new connections in the
same network with which the current LPE could
share information if they are working on the same
assignment.

data acquisition
    conn [] = getConnectionsOnSameNetwork()
activation guards
    conn.size != 0
conditions
    conn.assignment.isStillValid()
actions
    foreach connection in conn []
      if(usersWorkOnSameAssignment(
      connection.assignemnt.ID))
        sendMessageToLPE(conn.ID, message)
        inviteOtherLPEforCollaborativeWork(
        connection.assignemnt.ID)
      end if
    end foreach
rules generation
    if(!activationGuard)
      createGMS002(conn.ID, conn.assignemnt.ID)
    end if
    cloneRule (GMS 001)
```

Figure 4. An example, in pseudo-code, of a Proactive Rule

A. Other fields of applications for LPEs

The previous case study indicated that LPEs could be used in education. In hospitals for example, LPEs could share information and create very accurate and useful reports for doctors. They could also be implemented in other domains, which have to handle big amounts of data coming from sensors, like other areas of medicine, transportation, engineering, aviation and many social networks platforms.

VI. A Short Implementation Overview

We are currently working on the implementation of LPEs for smartphones and tablets, with an Android Operating System, as they allow direct data exchange between devices which are on the same Wi-Fi network, without having an intermediate access point. Frameworks like Android's WiFi-P2P, SQLite™ [13] and ORMLite™ [14] will be used for creating the prototype. The Proactive Engine will run as a background service.

VII. Conclusion and Future Work

In this work-in-progress, we have identified and described a few of the important features and characteristics of a model that achieves to integrate proactive, self-adaptive and context-aware features into software systems. With the proposed model, the user is focusing more on how to interact

 ISBN: 978-1-61208-341-4

with the application and not how to manage and configure the system.

A. Challenges Ahead

Two of the most challenging points are to ensure the communication between the components of a LPE and to design proactive scenarios, while taking in account important factors like user mobility, different computing capabilities of various devices and privacy issues.

B. Future work

A case-study based evaluation will follow for validating all the characteristics of the presented model and for answering to some research questions such as whether or not the model is correctly providing routines in a context-adaptive manner, or if the parts of the model are really taking into account the user's preferences, or if the model has self-adaptive properties that allow it to modify its behavior.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges". ACM Transactions on Autonomous and Adaptive Systems, 2009, vol. 4, pp. 1-42.

[2] Y. Brun et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap" in Software Engineering for Self-Adaptive Systems, Lecture Notes In Computer Science, Springer, 2009, vol. 5525, pp. 48-70.

[3] S. Dobson et al., "A survey of autonomic communications". ACM Trans. Auton. Adapt. Syst., 2006, vol. 1, pp. 223-259.

[4] B.H.C. Cheng et al., "Engineering Self-Adaptive Systems through Feedback Loops" in Software Engineering for Self-Adaptive Systems, Lecture Notes In Computer Science, Springer, 2009, vol. 5525, pp. 1-26.

[5] D. Tennenhouse, "Proactive Computing". Communications of the ACM, 2000, vol. 43, issue 5, pp. 43-50.

[6] A. Oulasvirta and A. Salovaara, "Six modes of proactive resource management: a user-centric typology for proactive behaviors", in Proc. NordiCHI 2004, ACM Press, pp. 57-60.

[7] D. Zampunieris, "Implementation of a Proactive Learning Management System", in Proc. E-learn 2006, AACE Press, pp. 3145-3151.

[8] S. Coronado and D. Zampunieris, "Towards a proactive learning management system using early activity detection". In SITE08, AACE Publishing, 2008, vol. 1, pp. 306-311.

[9] R. Dobrican and D. Zampunieris, "Supporting collaborative learning inside communities of practive through proactive computing", in Proc. EDULEARN13, 2013, pp. 5824-5833.

[10] R. Dobrican, S. Reis, and D. Zampunieris, "Empirical Investigations on Community Building and Collaborative Work inside a LMS using Proactive Computing" in Proc. E-learn 2013, vol. 1, pp. 1840-1852.

[11] D. Shirnin, S. Reis, and D. Zampunieris, "Experimentation of Proactive Computing in Context Aware Systems: Case Study of Human-Computer Interactions in e-Learning Environment". IEEE CogSIMA, Feb. 2013, pp. 269-276.

[12] Moodle - Modular Object-Oriented Dynamic Learning Environment. [retrieved: April, 2014]. Available from: https://moodle.org/

[13] SQLite Framework. [retrieved: April, 2014]. Available from: http://www.sqlite.org/

[14] ORMLite - Lightweight Object Relational Mapping (ORM) Java Package. [retrieved: April, 2014]. Available from: http:// http://ormlite.com/

 ISBN: 978-1-61208-341-4

DAiSI—A Component Model and Decentralized Configuration Mechanism for Dynamic Adaptive Systems

Holger Klus
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
holger.klus@tu-clausthal.de

Andreas Rausch
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
andreas.rausch@tu-clausthal.de

***Abstract*— Dynamic adaptive systems are systems that change behavior according to the needs of the user during run time, based on context information. Since it is not feasible to develop these systems from scratch every time, a component model enabling dynamic adaptive systems is called for. Moreover, an infrastructure is required that is capable of wiring dynamic adaptive systems from a set of components in order to provide a dynamic and adaptive behavior to the user. In this paper we present just such an infrastructure or framework—called Dynamic Adaptive System Infrastructure (DAiSI). The focus of the paper is on the underlying component model and the decentralized configuration mechanism. We will present an example scenario illustrating the adaptation capabilities of the framework we introduce.**

Keywords-dynamic adaptive systems; component model; component composition; adaptation; componentware; component container; decentralized configuration.

I. INTRODUCTION

Software-based systems pervade our daily life—at work as well as at home. Public administration or enterprise organizations can scarcely be managed without software-based systems. We come across devices executing software in nearly every household. The continuous increase in size and functionality of software systems has now made some of them among the most complex man-made systems ever devised [1].

In the last two decades the trend towards "everything, every time, everywhere" has been dramatically increased through a) smaller mobile devices with higher computation and communication capabilities, b) ubiquitous availability of the Internet (almost all devices are connected with the Internet and thereby connected with each other), and c) devices equipped with more and more connected, intelligent and sophisticated sensors and actuators.

Nowadays these devices are increasingly used within an organically grown, heterogeneous, and dynamic IT environment. Users expect them not only to provide their primary services but also to collaborate autonomously with each other and thus to provide real added value. The challenge is therefore to provide software systems that are robust in the presence of increasing challenges such as change and complexity [2].

The reasons for the steady increase in complexity are twofold: On the one hand, the set of requirements imposed on software systems is becoming larger and larger as the extrinsic complexity increases, in the form of, for example, additional functionality and variability. In addition, the structures of software systems—in terms of size, scope, distribution and networking of the system among other things—are themselves becoming more complex, which leads to an increase in the intrinsic complexity of the system.

Change is inherent, both in the changing needs of users and in the changes which take place in the operational environment of the system. Hence it is essential that our systems be able to adapt as necessary to continue to satisfy user expectations and environmental changes in terms of an evolutionary change. Dynamic change, in contrast to evolutionary change, occurs while the system is operational. Dynamic change requires that the system adapt at run time.

Since the complexity and change may not permit human intervention, we must plan for automated management of adaptation. The systems themselves must be capable of determining what system change is required, and in initiating and managing the change process wherever possible. This is the aim of self-managed systems.

Self-managed systems are those capable of adapting to the current context as required though self-configuration, self-healing, self-monitoring, self-tuning, and so on. These are also referred to as self-x, autonomic systems. We call them dynamic adaptive systems.

Providing dynamic adaptive systems is a great challenge in software engineering [2]. In order to provide dynamic adaptive systems, the activities of classical development approaches have to be partially or completely moved from development time to run time. For instance, devices and software components can be attached to a dynamic adaptive system at any time. Consequently, devices and software components can be removed from the dynamic adaptive system or they can fail as the result of a defect. Hence, for dynamic adaptive systems, system integration takes place during run time.

To support the development of dynamic adaptive systems a couple of infrastructures and frameworks have been developed, as discussed in a related work section, Section 2. In our research group we have also developed a framework for dynamic adaptive (and distributed) systems, called DAiSI

 ISBN: 978-1-61208-341-4

(Dynamic Adaptive System Infrastructure). The first version of DAiSI was implemented and published in 2006/07 [15], [10], [14] , [11]. Based on the DAiSI framework a couple of dynamic adaptive systems (research and industrial demonstrators) were developed and evaluated within the following domains: assisted sport training systems [3], emergency management systems [7], [9], assisted living systems for elderly people [8], [10], intelligent beer dispensing systems [5], [6], and airport baggage management system [12], [13], [11]. All of these systems were exhibited at CeBIT, such as [4]. Some of them were successfully transformed into products, for instance [5] and [6].

Based on the evaluation results a couple of drawbacks were identified. I) DAiSI's component model was not able to handle manage service cardinalities, such as exclusive and shared use of a specific service or service reference sets. Most of the applications realized needed service cardinalities. Due to the absence of service cardinalities we had to create workarounds. II) DAiSI's dynamic configuration mechanism was realized as a centralized component. The centralized configuration component was easy to implement but obviously it turned out to be a bottleneck.

For that reasons we have developed and implemented an improved version of the DAiSI framework. It contains a sophisticated component model including service cardinalities and a decentralized system configuration mechanism. In this paper the new version of the DAiSI framework will be presented.

The rest of the paper is structured as follows: After a short description of the related work we provide an overview of the DAiSI framework. In the following three subsections we will introduce DAiSI's main essential: a domain model, an adaptive component model, and a decentralized dynamic configuration mechanism. Then we describe a small sample application to illustrate the decentralized dynamic configuration mechanism of the adaptive components. A short conclusion will round the paper up.

II. RELATED WORK

Component-based software development, component models and component frameworks provide a solid approach to support evolutionary changes to systems. Components are the units of deployment and integration. During design time components may be added or removed from a system [16].

However, dynamic changes, e.g. adding or removing components from a system during run time is not direct support. Service-oriented approaches promise a more flexible approach for dynamic changes. Service users query for services within a service directory. Once they have found the corresponding service they can dynamically connect themselves to the service [17].

Unfortunately in service-oriented approaches the components are responsible for the dynamic adaptive behavior. They have to query for the proper services, verify that the services fit the ones they are looking for and connect themselves to the corresponding services. For that reason a couple of frameworks have been developed. Those frameworks support the component configuration during run time and thereby form dynamic adaptive systems. CONIC and REX provide a description technique to describe an initial system configuration and system adaptions during run time [18], [19].

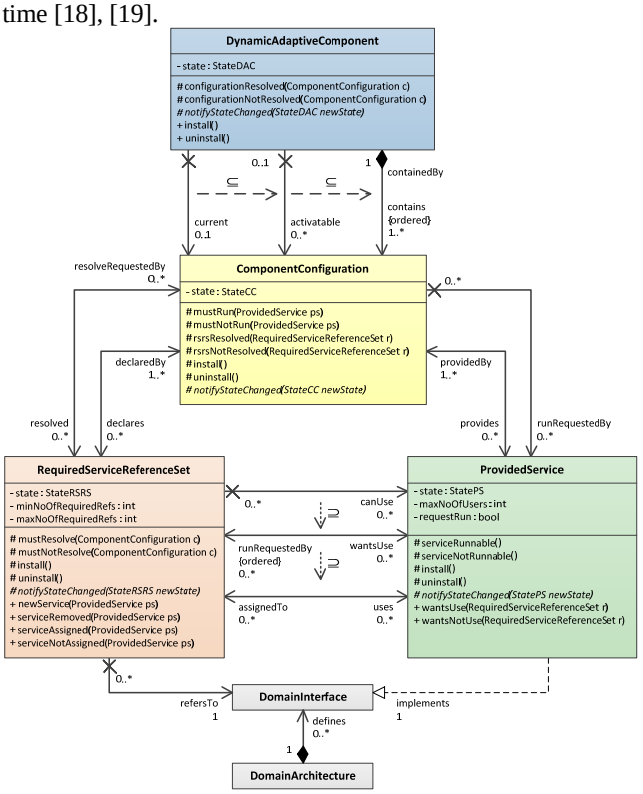


Figure 1. Core elements of the DAiSI framework.

Current frameworks such as ProAdapt [20] and Config.NETServices [21] have a more generic adaption and configuration mechanism. Components that were not known during the design-time of the system, are added and removed from the dynamic adaptive system during run time. Therefore a generic component configuration mechanism is provided by the framework. As with our first version of the DAiSI framework, these frameworks are based on a centralized configuration mechanism. Moreover the underlying component model is restricted—for instance the exclusive usage of services cannot be described.

III. DAISI – DYNAMIC ADAPTIVE SYSTEM INFRASTRUCTURE

Our approach for self-organizing systems is based on a specific framework called DAiSI [15], [10], [14] , [11]. DAiSI consists of three main parts or elements: a domain model, an adaptive component model, and a decentralized dynamic configuration mechanism. All three will be introduced at a glance in the following section. The three elements and their relationship to each other are depicted in Figure 1 using a UML class diagram. Note, a complete description of the DAiSI framework can be found in [22].

 ISBN: 978-1-61208-341-4

A. Domain Model

As in other domains, such as the network domain, physical connectors (like the RJ 45 connector) and their pin configurations are standard and well known by all component vendors. A similar situation can be found in the operating system domain: The interface for printer drivers is standardized and published by the operating system vendor. Third-party printer vendors adhere to this interface specification to create printer drivers that are plugged into the operating system during run time.

The same principle is used in the DAiSI framework: The domain model contains standardized and broadly accepted interfaces in the domain. The domain model defines the basic notions and concepts of the domain shared by all components. This means the domain model provides the foundation for the dynamic configuration of the adaptive system and the available components.

The domain model, as shown in Figure 1, consists of the *DomainInterface* and *DomainArchitecture* classes. The domain model itself is represented by an instance of the *DomainArchitecture* class. A domain model contains a set of domain interfaces, represented by an instance of the class *DomainInterface*.

Domain interfaces contain syntactical information like method signatures or datatypes occuring in the interfaces. In addition they may also contain a behavioral specification of the interface following the design by contract approach, for instance using pre- and postconditions and invariants to describe the functional behavior of a domain interface [9].

Usually components need services from other components to provide their own service within the dynamic adaptive system. To indicate which services a component provides and requires it refers to the corresponding *DomainInterface*. As components providing services and components requiring services refer to the same domain interface description DAiSI is able to identify those and bind these components together during run time.

Using simple domain interface descriptions the correctness of the binding can only be guaranteed on a syntactical level. Once the domain interface descriptions contain additional information about the functional behavior, the correctness of the binding can also be guaranteed on the behavioral level. Therefore we have developed a sophistic approach based on run-time testing. Further information of DAiSI's solution to guarantee functional correctness of dynamic adaptive systems during run time can be found in [9], [23].

B. Adaptive Component Model

Each component in the system is represented by the *DynamicAdaptiveComponent* class. Each component may provide services to other components or use services, provided by other components. The services a component provides are represented by the *ProvidedService* class. The services a component requires are specified by the *RequiredServiceReferenceSet* class, where each instance represents a set of required services for exactly one domain interface. The *ComponentConfiguration* class of the component model represents a mapping between services required and provided. If all the required services of a component configuration are available, the provided services of that component configuration can in turn be provided to other components. In the following subsections the individual parts of the component model are introduced in more detail. Afterwards, the interplay of these parts during the configuration process will be explained.

1) Dynamic Adaptive components

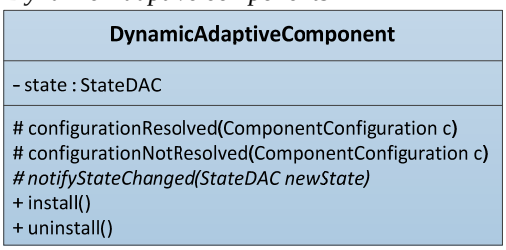


Figure 2. DynamicAdaptiveComponent class.

Each component instance within the system is represented by an instance of the class DynamicAdaptiveComponent, see Figure 2. By calling the install or uninstall methods, a component is, respectively, published or removed from the system. If install is called, all other parts of that component are informed by calling the trigger install. The framework then starts trying to resolve dependencies on other components in order to run ProvidedServices and provide them to other components within the system. Each DynamicAdaptiveComponent realizes a state machine, as shown in Figure 3 whose current state is stored in a variable called state.

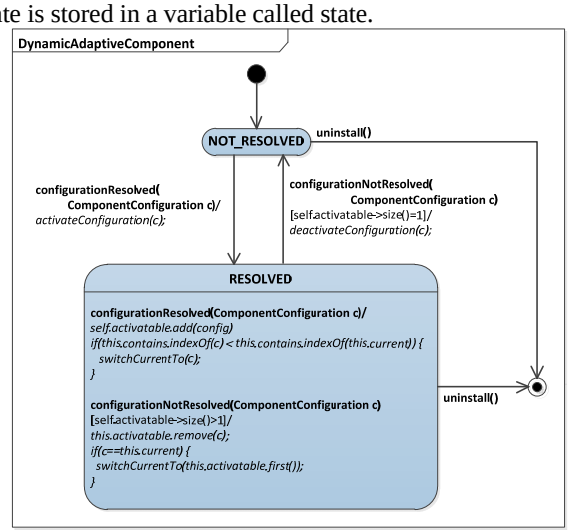


Figure 3. State machine - DynamicAdaptiveComponent class.

Two states are distinguished for *DynamicAdaptiveComponent*, namely RESOLVED and NOT_RESOLVED. In the beginning a component is in the NOT_RESOLVED state. If, for a single *ComponentConfiguration*, all dependencies to services of other components are resolved, the trigger *configurationResolved* of *DynamicAdaptiveComponent* is called and the state machine switches to state RESOLVED.

 ISBN: 978-1-61208-341-4

Every time a state transition takes place, the abstract method, *notifyStateChanged*, is called. A component developer can override this method in order to react to certain state transitions, e.g. by showing or fading out a graphical user interface.

2) Component Configuration

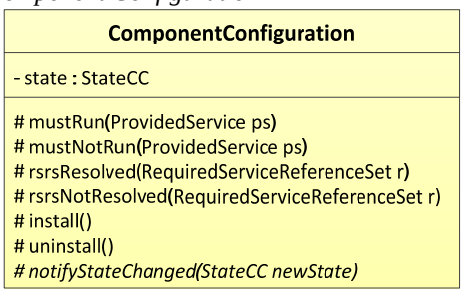


Figure 4. ComponentConfiguration class.

Each component defines at least one *ComponentConfiguration*. Figure 4 shows the corresponding class diagram for *ComponentConfiguration.* The defined *ComponentConfigurations* are connected to a component by the association contains. Each *ComponentConfiguration* represents a mapping between a set of required and provided services. If all services required by a *ComponentConfiguration* are available, the corresponding provided services can be provided to other components. That configuration is then marked as *activatable*. In case a component has more than one *ComponentConfiguration*, an order must be defined by the component developer. During run time, at most one *ComponentConfiguration* can be active. That one is then marked as current and only those provided services are executed that are connected to *ComponentConfiguration*, which is marked as current.

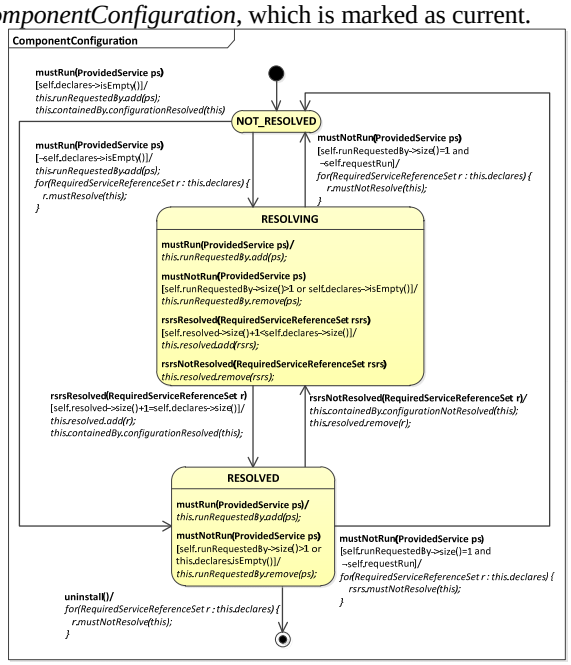


Figure 5. State machine - ComponentConfiguration class.

Each *ComponentConfiguration* realizes a state machine, as shown in Figure 5, with three states, namely NOT_RESOLVED, RESOLVING and RESOLVED. If a *ProvidedService* has to be executed (e.g. because another component needs it), the trigger mustRun of *ComponentConfiguration* is called. Afterwards the trigger *mustResolve* is called at each *RequiredServiceReferenceSet* in order to initiate the resolving of dependencies to other components. A *RequiredServiceReferenceSet* informs the *ComponentConfiguration* of the current status of the dependency resolution by calling the triggers *rsrsResolved* and *rsrsNotResolved*. A *ComponentConfiguration* is in RESOLVED state if the dependencies of all required services are resolved, i.e. all connected *RequiredServiceReferenceSets* have called the trigger *rsrsResolved*. The *ComponentConfiguration* in turn calls *configurationResolved* to inform the *DynamicAdaptiveComponent*.

3) Provided Service

A component's provided services are represented by the class *ProvidedService* shown in the class diagram in Figure 6. Each one implements exactly one domain interface. For each *ProvidedService* the number of service users who are allowed to use the service in parallel can be specified. This is done by setting the variable *maxNoOfUsers* to the required value. In our component model, a service is executed for only two reasons. The first reason is that there exist one or more components that want to use that service. Requests for service usage can be placed by calling the method *wantsUse*, or *wantsNotUse* if the usage request has become invalid. If there is a usage request for a *ProvidedService*, the connected *ComponentConfigurations* are informed by calling the trigger *mustRun*. The second reason that a service might have to be executed is that it provides some kind of direct benefit for end users. A component developer can set the flag *requestRun* in this case (e.g. because the service realizes a graphical user interface).

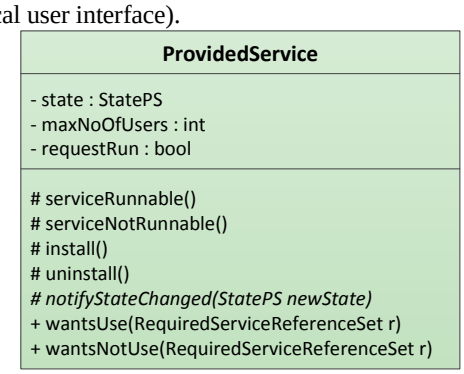


Figure 6. ProvidedService class.

A *ProvidedService* realizes a state machine with three states namely NOT_RUNNING, RUNNABLE and RUNNING, as illustrated in Figure 7. A service is in RUNNABLE state if it is exclusively connected to *ComponentConfigurations* whose dependendies are resolved but none of them is marked as current. This is the case for a *ComponentConfiguration* that has higher priority and that is

marked as *activatable*. However, a service is in RUNNING state if it is connected to a *ComponentConfiguration* which is marked as *current*. If a *ComponentConfiguration* becomes current, all connected *ProvidedServices* are informed by calling the *serviceRunnable* trigger.

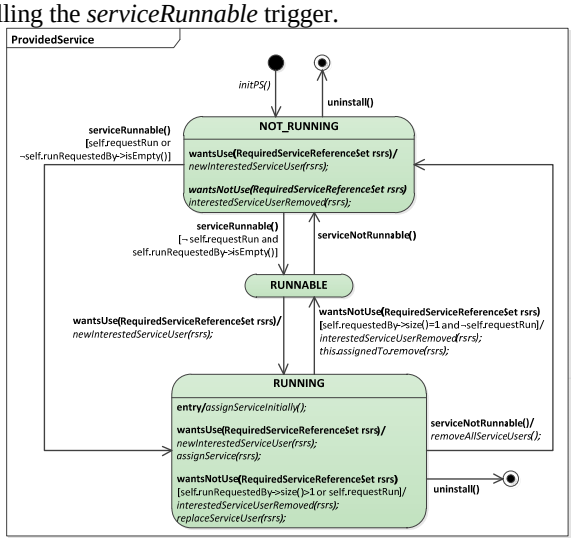


Figure 7. State machine - ProvidedService class.

4) Required Service Reference Set

A component may need functionality provided by other components in the system. In our component model those dependencies are specified with the *RequiredServiceReferenceSet* class, shown in Figure 8. Each instance of *RequiredServiceReferenceSet* represents dependencies on a set of services that implement the same domain interface. That domain interface is specified by the association, *refersTo*. A component representing a trainer for example may define a *RequiredServiceReferenceSet* that refers to a domain interface called *IAthlete* in order to get access to the training data of athletes. The minimum and maximum number of required references to services can be specified by setting the variables *minNoOfRequiredRefs* and *maxNoOfRequiredRefs*.

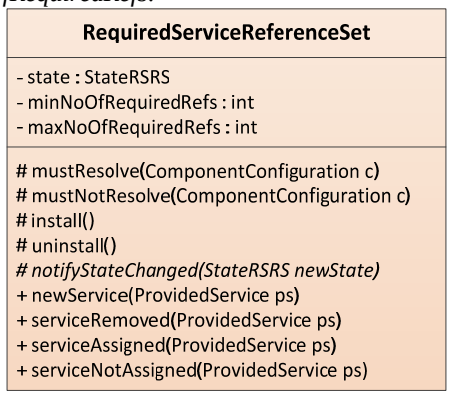


Figure 8. RequiredServiceReferenceSet class.

A *RequiredServiceReferenceSet* realizes a state machine with three states, namely NOT_RESOLVED, RESOLVING and RESOLVED. Figure 9 visualizes this state machine. As soon as there is a request for resolving dependencies, the state switches to RESOLVED or RESOLVING, depending on the value of *minNoOfRequiredRefs*. If it is zero, then the requirements are fulfilled and it can switch directly to RESOLVED. A request for dependency resolution is placed by calling the *mustResolve* trigger.

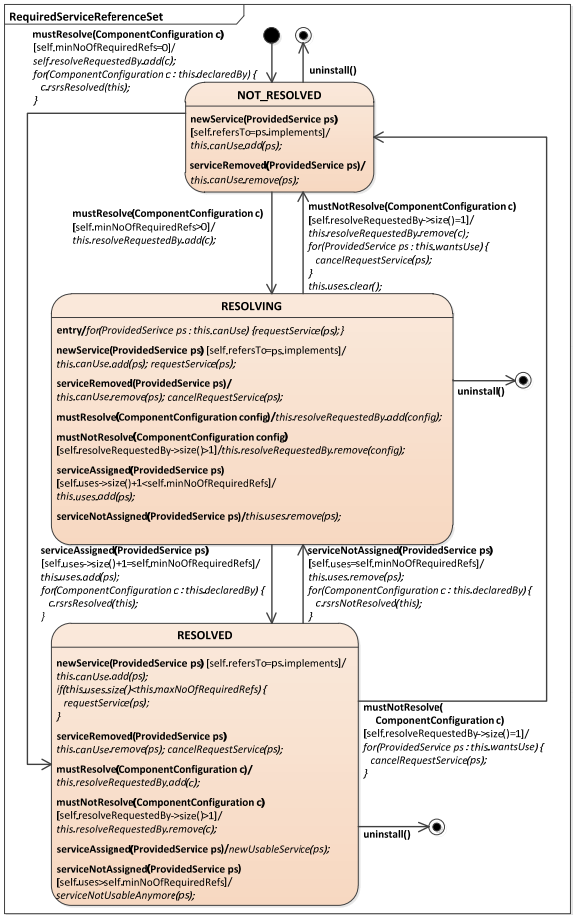


Figure 9. State machine - RequiredServiceReferenceSet class.

5) Notation for DAiSI Components

To describe DAiSI components we use a compact notation, illustrated in Figure 10. Provided services are notated as circles, required services as semicircles, component configurations are depicted as crossbars, and the component itself is represented by a rectangle. Provided services that are intended to be activated (flag *requestRun* is true) are shown as a black circle.

 ISBN: 978-1-61208-341-4

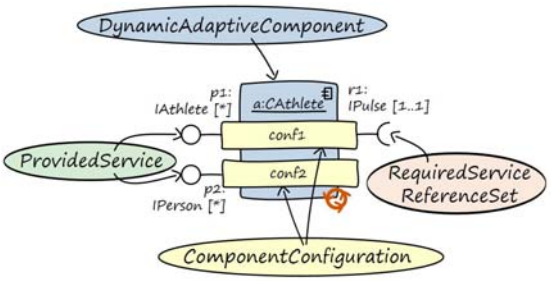


Figure 10. Notation for DAiSI components.

The component depicted in Figure 10 thus specifies two component configurations. The first requires exactly one service, which implements the *DomainInterface IPulse*. If such a service is available, the service variable p_1 of type *IAthlete* can in turn be provided to other components in the system. If no pulse service is available, the second configuration can still be activated because that one defines no dependencies to other services. In that case, the athlete component provides the service variable p_2 to other components.

C. Decentralized Dynamic Configuration Mechanism

There exist three types of relations between *RequiredServiceReferenceSets* and *ProvidedServices*, represented by the associations *canUse*, *wantsUse* and uses. The set of services that implement the domain interface referred by the *RequiredServiceReferenceSet* is represented by *canUse*. Note, this only guarantees a syntactically correct binding. In [9] and [23] we have shown how this approach can be extended to guarantee functional-behaviorally correct binding as well during run time using a run-time testing approach.

The *wantsUse* set holds references to those services for which a usage request has been placed by calling *wantsUse*. And the uses set contains references to those services which are currently in use by the component or by *RequiredServiceReferenceSet*.

Each time a new service becomes available in the system, the newService method is called with a reference to the service as parameter. The new service is added to all *canUse* sets, if the corresponding *RequiredServiceReferenceSet* refers to the same *DomainInterface* as the *ProvidedServices*. If there is a request for dependency resolution (by a call of the *mustResolve* trigger), usage requests are placed at the services in *canUse* by calling *wantsUse* and those service references are copied to the *wantsUse* set. *ProvidedServices*

The management of these three associations—*canUse*, *wantsUse* and uses—between *RequiredServiceReferenceSets* and *ProvidedServices* is handled by DAiSI's decentralized dynamic configuration mechanism. This configuration mechanism relays on the state machines, presented in the previous sections, of the corresponding classes in the DAiSI framework and their interaction. In the following section we will first describe the local configuration mechanism component and then the interaction between two components for inter-component configuration.

1) Local Configuration Mechanism

Assume a given component as shown in Figure 11. The component t of type *CTrainer* has a single configuration. It provides a service of type *ITrainer* to the environment, which can be used by an arbitrary number of other components. The component requires zero to any number of references to services of type *IAthlete*.

Figure 11. CTrainer component.

The boolean flag *requestRun* is true for the service provided. Hence, DAiSI has to run the component and provide the service within the dynamic adaptive system to other components and to users. As the component requires zero reference to services of type *IAthlete*, DAiSI can run the component directly and thereby provide the component service to other components and users as shown in the sequence diagram in Figure 12.

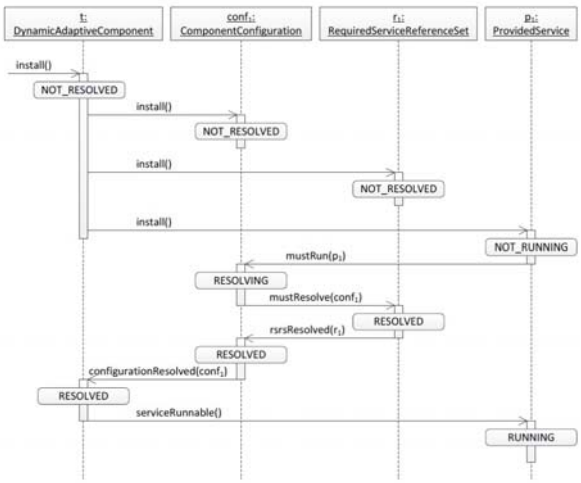


Figure 12. Local configuration mechanism component.

2) Inter-Component Configuration Mechanism

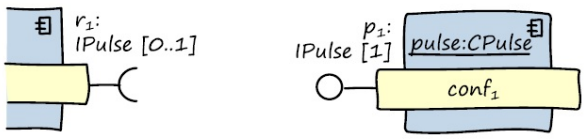


Figure 13. CAthlete and CPulse components.

Now assume two components: The *CAthlete* component, shown on the right hand side of Figure 13, requires zero or one reference to a service of type *IPulse*. The second component, *CPulse*, shown on the left hand side of Figure 13, provides a service of type *IPulse*. Note, this service can only be exclusively used by a single component.

 ISBN: 978-1-61208-341-4

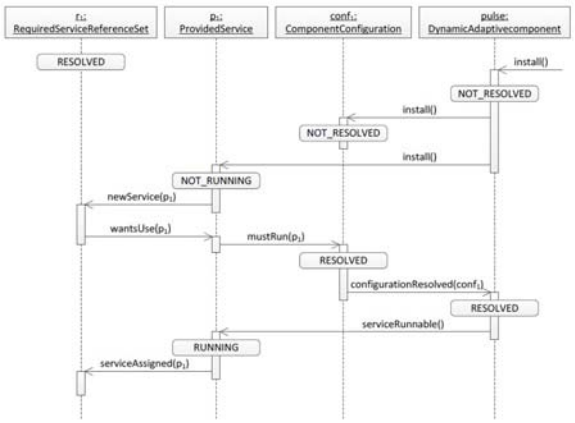


Figure 14. Inter-component configuration mechanism.

Once the *CPulse* component is installed or activated within the dynamic adaptive system, DAiSI integrates the new service in the canUse relationship of the *RequiredServiceReferenceSet* r_1 of the component *CAthlete*. Then DAiSI informs (calling the method *newService*) the *CAthlete* component that a new service that can be used is available as shown in Figure 14. DAiSI indicates that *CAthlete* wants to use this new service by adding this service in the set of services that *CAthlete* wants to use (set *wantUse* of *CAthlete*). Once the service runs it is assigned to the *CAthlete* component which can use the service from now on (added to the set uses of *CAthlete*).

IV. SAMPLE APPLICATION – SMART BIATHLON TRAINING SYSTEM

As already mentioned we have realized and used a couple of dynamic adaptive systems based on DAiSI. One of the first domains for which we developed dynamic adaptive systems was training systems for athletes. For that reason we have chosen this domain to implement the first dynamic adaptive system on top of the new DAiSI version.

A. Domain Model

In the desired dynamic adaptive system, athletes (*IAthlete*) and trainers (*ITrainer*) can supervise the pulse (*IPulse*) of the athlete (see Figure 15). Moreover athletes my use ski sticks (*IStick*), which have gyro sensors. Once connected with the sticks the athlete as well as the trainer can monitor the technically appropriate use of the sticks during skiing for the required skiing style. Once the biathlete has reached a shooting line (*IShootingLine*) he is allowed to use the shooting line only if a superviser is available (*ISupervisor*).

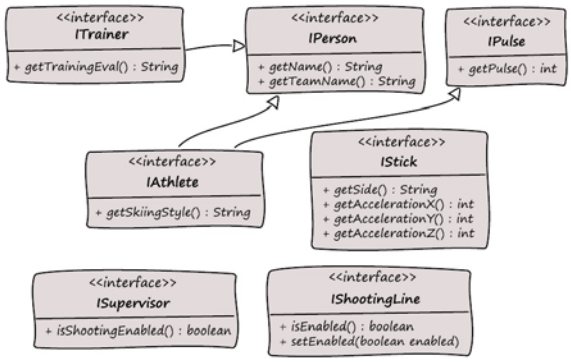


Figure 15. Domain model - "Smart Biathlon Training System".

B. Available Components

For a simple version of the system only three component types have been realized (see Figure 16): *CPulse*, *CAthlete*, and *CTrainer*. Note that additional components have been realized and evaluated for more sophisticated systems. For the purposes of this paper we only use these three components to show the decentralized configuration mechanism.

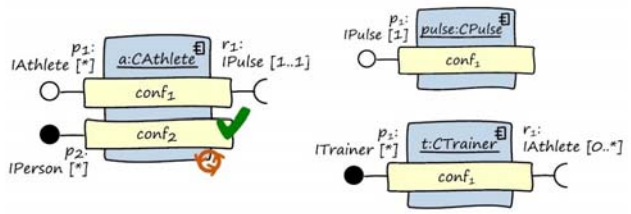


Figure 16. Adaptive components: CPulse, CAthlete, CTrainer.

The *CPulse* component provides an exclusive usable service *IPulse* and requires no other services from the dynamic adaptive system. The *CAthlete* component provides two services: *IPerson* and *IAthlete*. In $conf_2$ it provides the service, *IPerson*, which has the flag, *requestRun*, and requires no service from the environment. In conf1 it provides the service, *IAthlete*, but therefore requires a service, *IPulse*. And finally the *CTrainer* component may supervise an arbitrary number of athletes and thus provides a corresponding number of *ITrainer* interfaces to the real trainer, supporting him with the online training information of the supervised athletes.

C. Decentralized Dynamic Configuration Mechanism

Assume the following situation in the dynamic adaptive system. The component, *CPulse*, is activated and the component, *CAthlete*, is activated, see Figure 17 As the *requestRun* flag of the provided service of $conf_2$ is set and no additional service references are needed, this configuration is activated and the service is provided within the dynamic adaptive system.

 ISBN: 978-1-61208-341-4

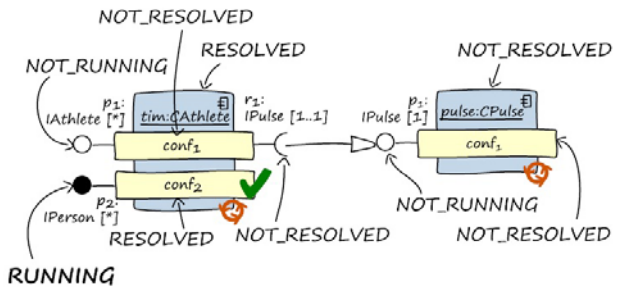
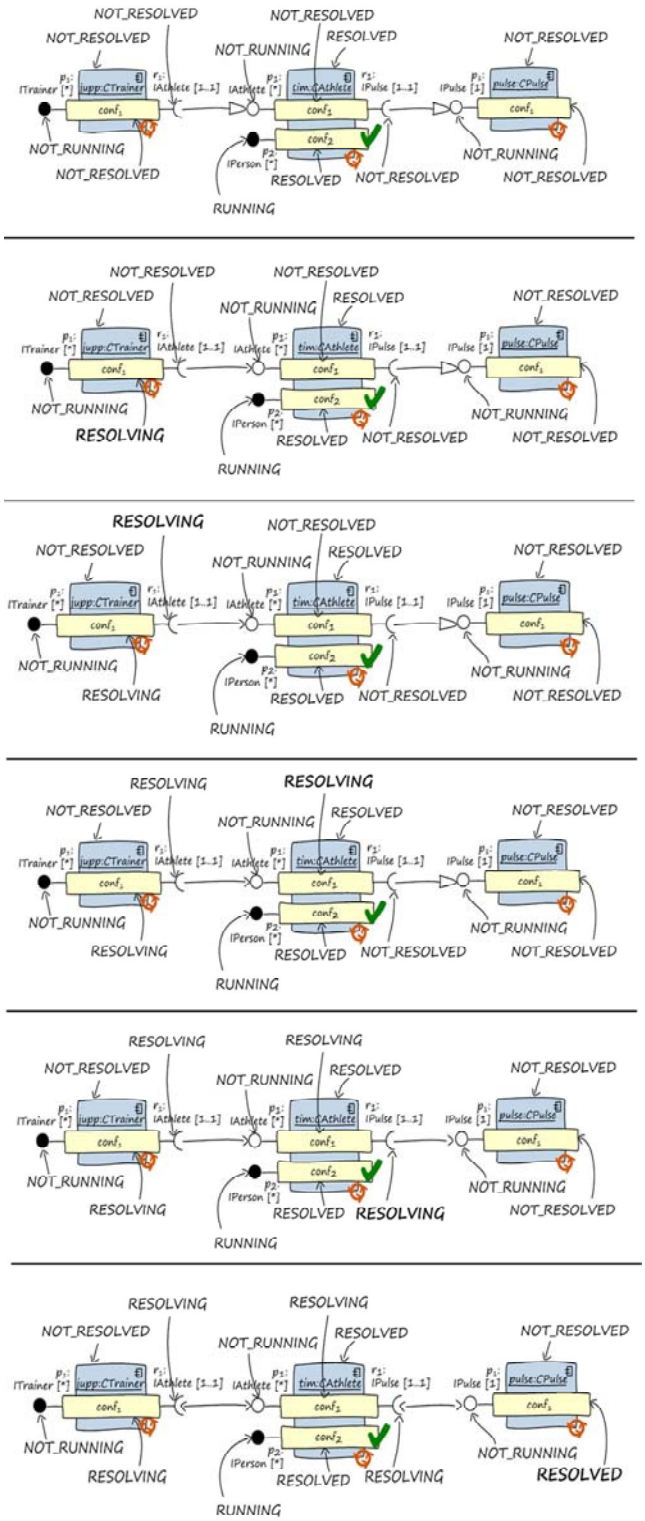


Figure 17. Initial situation in the Dynamic Adaptive System.

For the higher configuration, *conf*$_1$, *CAthlete* requires a reference to a service of type *IPulse*. The *CPulse* component is able to provide this service. As the provided service, *IAthlete*, of configuration conf1 of component *CAthlete* is not requested by any other component and has not set the *requestRun* flag, this higher configuration is not activated.

Figure 18 shows the following situation: A component, *CTrainer*, has been activated and integrated into our dynamic adaptive system. In the following the decentralized dynamic configuration mechanism is shown. Based on the interaction between the state machines of the adaptive components the dynamic adaptive system is reconfigured and the component is dynamically integrated into the system.

The configuration strategy is then as follows. Each service with *requestRun* flag set—in Figure 18 the new service *ITrainer* of the *CTrainer* component—resolves the required services transitively from the root to the leaf. Once all required services are resolved these services are activated (RUNNING) from the leaf to the root. If not all required services were resolvable, the resolved services are set back to NOT_RESOLVED. This allows other services to resolve these services.



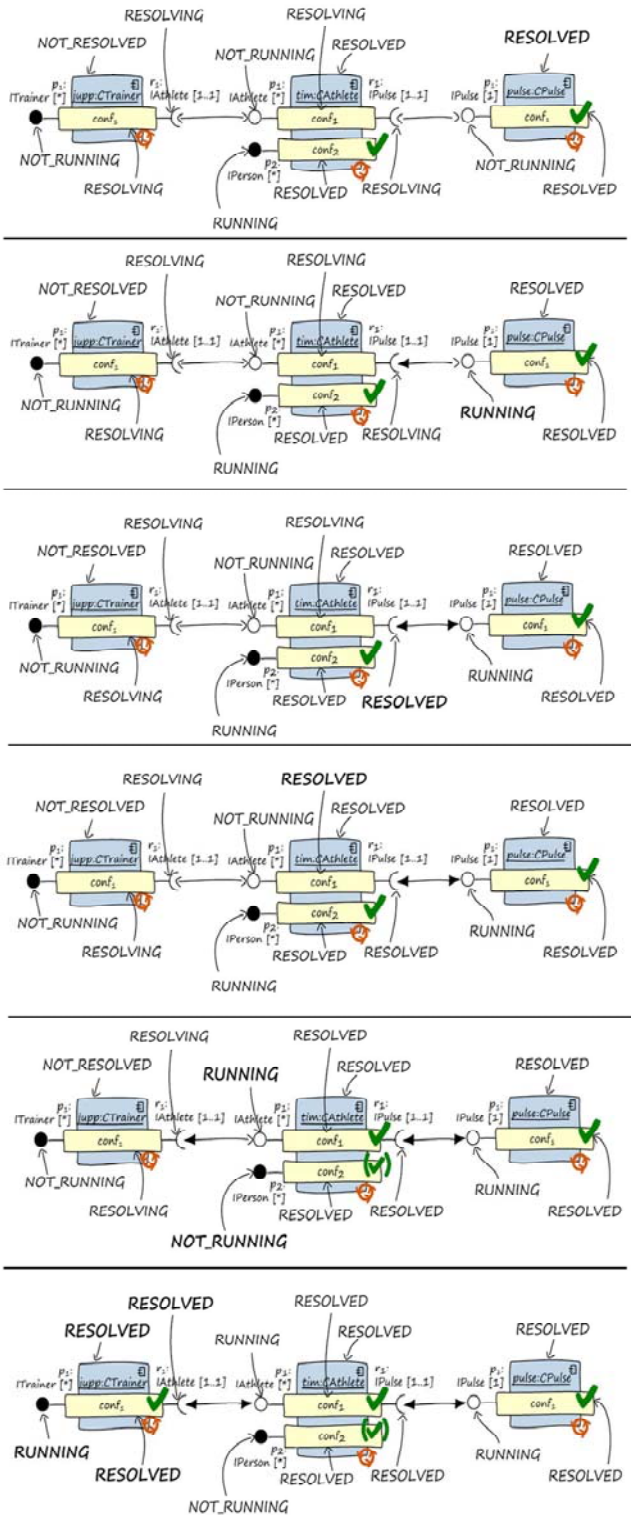


Figure 18. Step-by-Step decentralized dynamic configuration of the Smart Biathlon Training System.

V. CONCLUSION

The DAiSI approach is that a developer does not have to implement a whole dynamic adaptive system on his own. Instead the developer can develop one or more components for a specific domain. This is only possible if a domain model is available as described. This domain model has to define the interfaces between the adaptive components of the dynamic adaptive system in the specific domain.

Based on this, the developer can develop even a single component and define which interfaces from the domain architecture are required or provided in the different configurations of this component. Moreover one can develop mock-up components providing the required interfaces in order to test the new component during development.

To support the component development DAiSI comes with two implementation frameworks. These frameworks provide several helper classes enabling a quick implementation of dynamic adaptive systems in Java as well as in C++, concentrating on the functional features of the component to be developed. DAiSI-based dynamic adaptive systems can be distributed across various machines. DAiSI is also able to establish dynamic adaptive systems across language barriers—Java- and C++-based DAiSI components can be linked together through DAiSI to form a dynamic adaptive system.

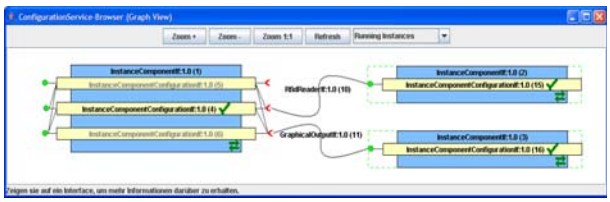


Figure 19. DAiSI Dynamic Adaptive System Monitor.

In order to monitor and debug a DAiSI-based dynamic adaptive system during development, the developer may use the so called “Dynamic Adaptive System Configuration Browser.” This allows to view the internal structure of the dynamic adaptive system in a graphical tree view.

As discussed in the introduction, DAiSI was used to realize and evaluate a couple of different applications. This allowed two main drawbacks of DAiSI to be identified: lack of service cardinalities and the centralized configuration mechanism.

In this paper we have shown DAiSI’s new component model supporting service cardinalities and the new decentralized dynamic configuration mechanism. A first dynamic adaptive system has been successfully implemented in the assisted sports training domain.

Consequently, further systems will be realized based on the new DAiSI version. Additional research is required to establish concepts to provide a proper balance between controllability of the system’s applications and the autonomy of the system components participating in these applications.

 ISBN: 978-1-61208-341-4

REFERENCES

[1] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems—The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon, Tech. Rep., June 2006.

[2] J. Kramer and J. Magee. A rigorous architectural approach to adaptive software engineering. Journal of Computer Science and Technology, 24(2):183{188, 2009}.

[3] T. Jaitner, M. Trapp, D. Niebuhr, and J. Koch, "Indoor simulation of team training in cycling," in ISEA 2006, E. Moritz and S. Haake, Eds. Munich, Germany: Springer, Jul. 2006, pp. 103–108.

[4] Emergency assistance system, Webpage of the cebit exhibit 2009, http://www2.in.tu-clausthal.de/~Rettungsassistenzsystem/, accessed 2014

[5] Intelligent beer dispensing system, Webpage of the cebit exhibit 2010", http://www2.in.tu-clausthal.de/~smartschank/systembeschreibung.php, Online; accessed 2014

[6] DIRMEIER SmartSchank, Intelligent Beer Dispensing System, DIRMEIER GmbH, http://www.dirmeier.de/DIRMEIER-0-0-0-1-1-1.htm, Online; accessed 2014

[7] A. Rausch, D. Niebuhr, M. Schindler, and D. Herrling. Emergency Management System. In Proceedings of the International Conference on Pervasive Services 2009 (ICSP 2009), 2009.

[8] Bilateral German-Hungarian Collaboration Project on Ambient Intelligent Systems. http://www.belami-project.hu/~micaz/belamiproject/history/part1. Online; accessed 2014.

[9] D. Niebuhr and A. Rausch. Guaranteeing Correctness of Component Bindings in Dynamic Adaptive Systems based on run-time Testing. In Proceedings of the 4th Workshop on Services Integration in Pervasive Environments (SIPE 09) at the International Conference on Pervasive Services 2009 (ICSP 2009). 2009.

[10] H. Klus, D. Niebuhr, and A. Rausch. A Component Model for Dynamic Adaptive Systems. In Proceedings of the International Workshop on Engineering of software services for pervasive environments (ESSPE 2007), 2007.

[11] H. Klus, D. Niebuhr, and A. Rausch. Dependable and Usage-Aware Service Binding. In Proceedings of the third International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2011), 2011.

[12] A. Rausch and D. Niebuhr. ECas News Journal, DemSy—A Scenario for an Integrated Demonstrator in a Smart City. 2010.

[13] C. Deiters, M. Köster, S. Lange, S. Lützel, B. Mokbel, C. Mumme, and D. Niebuhr, NTH computer science report, DemSy—A Scenario for an Integrated Demonstrator in a SmartCity. 2010.

[14] D. Niebuhr, H. Klus, M. Anastasopoulos, J. Koch, O. Weiß, and A. Rausch. DAiSI—Dynamic Adaptive System Infrastructure. Technical Report Fraunhofer IESE, 2007.

[15] M. Anastasopoulos, H. Klus, J. Koch, D. Niebuhr, and E. Werkman. DoAmI—A Middleware Platform facilitating (Re-)configuration in Ubiquitous Systems. In Proceedings of the Workshop on System Support for Ubiquitous Computing (UbiSys). 2006.

[16] C. Szyperski. Component Software. Addison Wesley Publishing Company. 2002.

[17] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In: Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE 2003). 10-12 December, Rome, Italy: IEEE Computer Society Press, 2003, S. 3–12.

[18] J. Magee, J. Kramer, and M. Sloman. Constructing Distributed Systems in Conic. In: IEEE Transactions on Software Engineering 15 (1989), Nr. 6, S. 663–675

[19] J. Kramer. Configuration Programming: A Framework for the Development of Distributable Systems. In: Proceedings of IEEE International Conference on Computer Systems and Software Engineering (COMPEURO 90). 8-10 May 1990, Tel-Aviv, Israel: IEEE Computer Society Press, 1990. ISBN 0818620412, S. 374–384

[20] R. R. Aschoff, and A. Zisman. Proactive adaptation of service composition. In: H. A. Müller, L. Baresi (Hrsg.): Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12): Zürich, Switzerland, June 4-5, 2012. Los Alamitos, California: IEEE Computer Society Press, 2012, S. 1–10

[21] A. Rasche, A. Polze. Configuration and Dynamic Reconfiguration of Component-based Applications with Microsoft .NET. In: Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003). 14-16 May 2003, Hakodate, Hokkaido, Japan: IEEE Computer Society Press, 2003. ISBN 0-7695-1928-8, S. 164–171

[22] H. Klus. Anwendungsarchitektur-konforme Konfiguration selbstorganisierender Softwaresysteme, Ph.D. Thesis, Technische Universität Clausthal, 2013.

[23] D. Niebuhr. Dependable Dynamic Adaptive Systems: Approach, Model, and Infrastructure. Clausthal-Zellerfeld, Technische Universität Clausthal, Institut für Informatik. Dissertation. 2010

 ISBN: 978-1-61208-341-4

An Adaptive Middleware for Near-Time Processing of Bulk Data

Martin Swientek
Paul Dowland

School of Computing and Mathematics
Plymouth University
Plymouth, UK
e-mail: {martin.swientek, p.dowland}@plymouth.ac.uk

Bernhard Humm
Udo Bleimann

Department of Computer Science
University of Applied Sciences Darmstadt
Darmstadt, Germany
e-mail: {bernhard.humm, udo.bleimann}@h-da.de

***Abstract*—The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change over time. In this paper, we introduce the concept of a middleware that is able to adapt its processing type fluently between batch processing and single-event processing. By adjusting the data granularity at runtime, the system is able to minimise the end-to-end latency for different load scenarios.**

Keywords–adaptive middleware; message aggregation; latency; throughput

I. INTRODUCTION

Enterprise Systems like customer-billing systems or financial transaction systems are required to process large volumes of data in a fixed period of time. For example, a billing system for a large telecommunication provider has to process more than 1 million bills per day. Those systems are increasingly required to also provide near-time processing of data to support new service offerings.

Traditionally, enterprise systems for bulk data processing are implemented as batch processing systems [1]. Batch processing delivers high throughput but cannot provide near-time processing of data, that is the end-to-end latency of such a system is high. End-to-end latency refers to the period of time that it takes for a business process, implemented by multiple subsystems, to process a single business event. For example, consider the following billing system of telecommunications provider:

- Customers are billed once per month
- Customers are partitioned in 30 billing groups
- The billing system processes 1 billing group per day, running 24h under full load.

In this case, the mean time for a call event to be billed by the billing system is 1/2 month. That is, the mean end-to-end latency of this system is 1/2 month.

A lower end-to-end latency can be achieved by using single-event processing, for example by utilizing a message-oriented middleware for the integration of the services that form the enterprise system. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. Therefore, message-based processing is usually not considered for building a system for bulk data processing requiring high throughput.

The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change during the lifespan of a system, either anticipated or not anticipated.

Additionally, enterprise systems often need to handle load peaks that occur infrequently. For example, think of a billing system with moderate load over most of the time, but there are certain events with very high load such as New Year's Eve. Most of the time, a low end-to-end latency of the system is preferable when the system faces moderate load. During the peak load, it is more important that the system can handle the load at all. A low end-to-end latency is not as important as an optimized maximum throughput in this situation.

In this paper, we propose a solution to this problem:

- We introduce the concept of a middleware that is able to adapt its processing type fluently between batch processing and single-event processing. By adjusting the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios. (Section III)

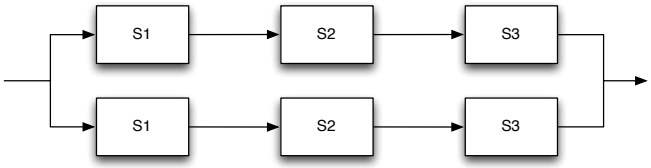
The remainder of this paper is organized as follows. Section II defines the considered type of system and the terms throughput and latency. The proposed middleware and the results of preliminary performance tests are presented in Section III. Section IV gives an overview of other work related to this reasearch. Finally, Section V concludes the paper and gives and outlook to the next steps of this research.

II. BACKGROUND

We consider a distributed system for bulk data processing consisting of several subsystems running on different nodes that together form a processing chain, that is, the output of subsystem S1 is the input of the next subsystem S2 and so on (see Figure 1a).

 ISBN: 978-1-61208-341-4

(a) Single processing line



(b) Parallel processing lines

Figure 1. A system consisting of several subsystems forming a processing chain

To facilitate parallel processing, the system can consist of several lines of subsystems with data beeing distributed among each line. For simplification, we consider a system with a single processing line in the remainder of this paper.

We discuss two processing types for this kind of system, batch processing and message-based processing.

A. Batch processing

The traditional operation paradigm of a system for bulk data processing is batch processing (see Figure 2). A batch processing system is an application that processes bulk data without user interaction. Input and output data is usually organized in records using a file- or database-based interface. In the case of a file-based interface, the application reads a record from the input file, processes it and writes the record to the output file.

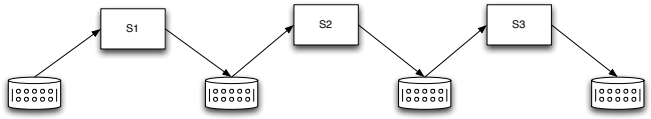


Figure 2. Batch processing

B. Message-base processing

Messaging facilitates the integration of heterogeneous applications using asynchronous communication. Applications are communicating with each other by sending messages (see Figure 3). A messaging server or message-oriented middleware handles the asynchronous exchange of messages including an appropriate transaction control [2].

Figure 3. Message-based processing

Message-based systems are able to provide near-time processing of data due to their lower latency compared with batch processing systems. The advantage of a lower latency comes with a performance cost in regard to a lower maximum throughput because of the additional overhead for each processed message. Every message needs, amongst others, to be serialized and deserialized, mapped between different protocols and routed to the appropriate receiving system.

C. End-to-end Latency vs. Maximum Throughput

Throughput and latency are performance metrics of a system. We are using the following definitions of maximum throughput and latency in this paper:

- **Maximum Throughput**
  The number of events the system is able to process in a fixed timeframe.
- **End-To-End Latency**
  The period of time between the occurrence of an event and its processing. End-to-end latency refers to the total latency of a complete business process implemented by multiple subsystems. The remainder of this paper focusses on end-to-end latency using the general term latency as an abbreviation.

Latency and maximum throughput are opposed to each other given a fixed amount of processing resources. High maximum throughput, as provided by batch processing, leads to high latency, which impedes near-time processing. On the other hand, low latency, as provided by a message-based system, cannot provide the maximum throughput needed for bulk data processing because of the additional overhead for each processed event.

III. An Adaptive Middleware for Near-Time Processing of Bulk Data

This section introduces the concept of an adaptive middleware which is able to adapt its processing type fluently between batch processing and single-event processing. It continuously monitors the load of the system and controls the message aggregation size. Depending on the current aggregation size, the middleware automatically chooses the appropriate service implementation and transport mechanism to further optimize the processing.

A. Middleware Components

Figure 4 shows the components of the middleware, that are based on the Enterprise Integration Patterns described by Hohpe et al. [3].

1) Aggregator: The Aggregator is a stateful filter which stores correlated messages until a set of messages is complete and sends this set to the next processing stage in the messaging route.

There are different options to aggregate messages, which can be implemented by the Aggregator:

- **No correlation**: Messages are aggregated in the order in which they are read from the input message queue. In this case, an optimized processing is not simply possible.

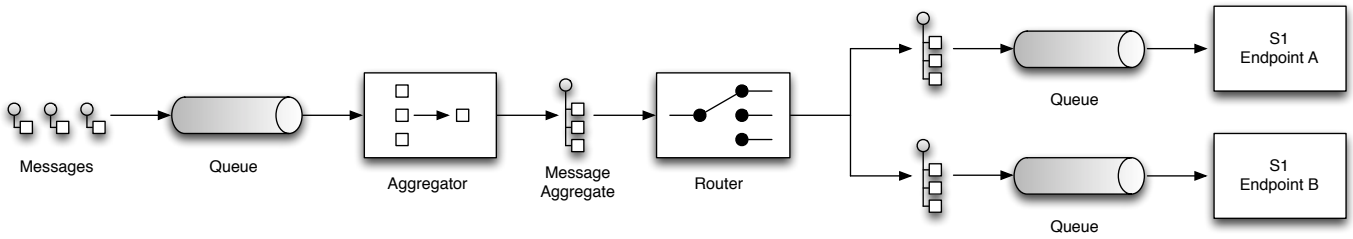


Figure 4. Components of the Adaptive Middleware. We are using the notation defined by [3]

- **Technical correlation:** Messages are aggregated by their technical properties, for example by message size or message format.
- **Business correlation**: Messages are aggregated by business rules, for example by customer segments or product segments.

2) Feedback Loop: To control the level of message aggregation at runtime, the middleware uses a closed feedback loop with the following properties (see Figure 5):

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.

Ultimately, we want to control the average end-to-end latency depending on the current load of the system. The change of queue size seems to be an appropriate quantity because it can be directly measured without a lag at each sampling interval, unlike the average end-to-end latency.

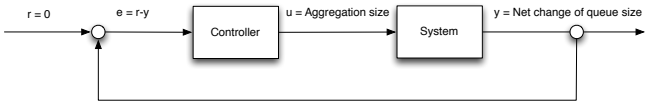


Figure 5. Feedback loop to control the aggregation size

The concrete architecture and tuning of the feedback loop and the controller is subject to our ongoing research.

3) Router: Depending on the size of the aggregated message, the Router routes the message to the appropriate service endpoint, which is either optimized for batch or single event processing.

When processing data in batches, especially when a batch contains correlated data, there are multiple ways to speed up the processing:

- To reduce I/O, data can be pre-loaded at the beginning of the batch job and held in memory.
- Storing calculated results for re-use in memory
- Use bulk database operations for reading and writing data

With high levels of message aggregation, it is not preferred to send the aggregated message payload itself over the message bus using Java Message Service (JMS) or SOAP. Instead, the message only contains a pointer to the data payload, which is transferred using File Transfer Protocol (FTP) or a shared database.

B. Prototype Implementation

To evaluate the proposed concepts of the adaptive middleware, we have implemented a prototype of a billing system using Apache Camel [4] as the messaging middleware.

Figure 6 shows the architecture of the prototype system.

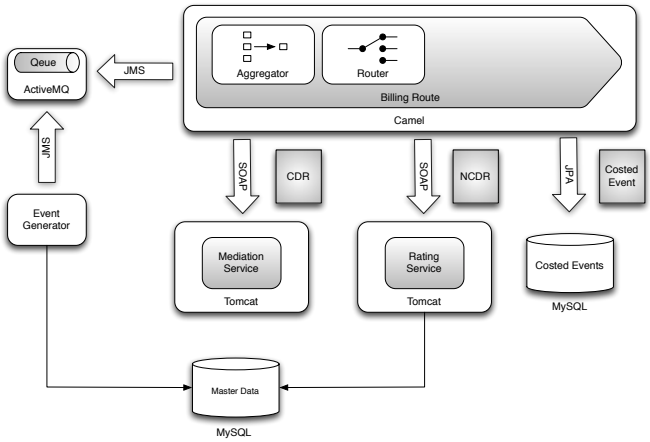


Figure 6. Architecture of the prototype system

Using this prototype, we have done some preliminary performance tests to examine the impact of message aggregation on latency and throughput. For each test, the input message queue has been pre-filled with 100.000 events. We have measured the total processing time and the processing time of each message with different static message aggregation sizes.

Figure 7 shows the impact of different aggregation sizes on the throughput of the messaging prototype. The throughput increases constantly for $1 < aggregation\_size <= 50$ with a maximum of 673 events per second with $aggregation\_size = 50$. Higher aggregation sizes than 50 do not further increase the throughput, it stays around 390 events per second.

The increased throughput achieved by increasing the aggregation size comes with the cost of a higher latency. Figure 8 shows the impact of different aggregation sizes on the 95th percentile latency of the messaging prototype.

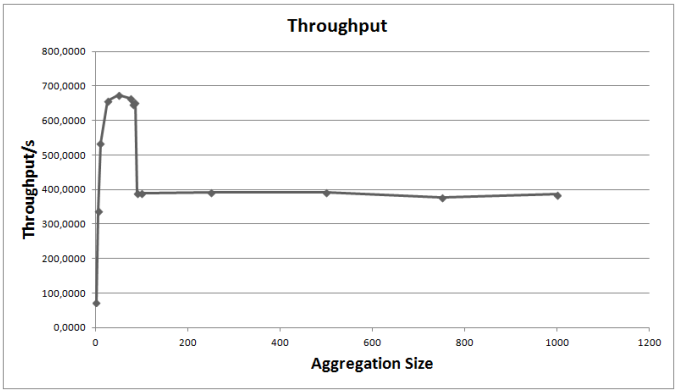


Figure 7. Impact of different aggregation sizes on throughput

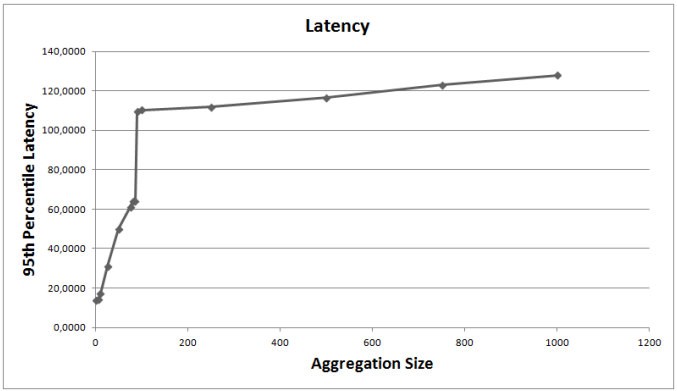


Figure 8. Impact of different aggregation sizes on latency

An aggregation size of 50, resulting in the maximum throughput of 673 events per seconds, shows a 95th percentile latency of about 68 seconds.

The results indicate that there is an optimal range for the aggregation size to control the throughput and latency of the system. Setting the aggregation size higher than a certain threshold leads to a throughput drop and latency gain. In case of our prototype, this threshold is between an aggregation size of 85 and 90. This threshold needs to be considered by the control strategy. We are currently investigating the detailed causes of this finding.

IV. Related Work

Research on messaging middleware currently focusses on Enterprise Services Bus (ESB) infrastructure. An ESB is an integration plattform that combines messaging, web services, data transformation and intelligent routing to connect multiple heterogeneous services [5]. It is a common middleware to implement the integration layer of an Service Oriented Architecture (SOA) and is available in numerous commercial and open-source packages.

Several research has been done to extend the static service composition and routing features of standard ESB implementations with dynamic capabilities decided at run-time, such as dynamic service composition [6], routing [7] [8] [9] and load balancing [10].

Work to manage and improve the Quality of Service (QoS) of ESB and service-based systems in general is mainly focussed on dynamic service composition and service selection based on monitored QoS metrics such as throughput, availability and response time [11]. Gonzaléz et al. [12] propose an adaptive ESB infrastructure to adress QoS issues in service-based systems which provides adaption strategies for response time degradation and service saturation, such as invoking an equivalent service, using previously stored information, distributing requests to equivalent services, load balancing and deferring service requests.

The adaption strategy of our middleware is to change the message aggregation size based on the current load of the system. Aggregating or batching of messages is a common approach to increase the throughput of a messaging system, for example to increase the throughput of total ordering protocols [13] [14] [15] [16].

A different solution to handle infrequent load spikes is to automatically instantiate additional server instances, as provided by current Platform as a Service (PaaS) offerings such as Amazon EC2 [17] or Google App Engine [18]. While scaling is a common approach to improve the performance of a system, it also leads to additional operational and possible license costs. Of course, our solution can be combined with these auto-scaling approaches.

V. Conclusion and Future Work

In this paper, we have presented a middleware that is able to adapt itself to changing load scenarios by fluently shifting the processing type between single event and batch processing. The middleware uses a closed feedback loop to control the end-to-end latency of the system by adjusting the level of message aggregation depending on the current load of the system. Determined by the aggregation size of a messsage, the middleware routes a message to appropriate service endpoints, which are optimized for either single-event or batch processing.

To evaluate the proposed middleware concepts, we have implemented a prototype system and performed preliminary performance tests. The tests show that throughput and latency of a messaging system depend on the level of data granularity and that the throughput can be increased by increasing the granularity of the processed messages.

Next steps of our research are the implementation of the proposed middleware including the evaluation and tuning of different controller architectures, performance evaluation of the proposed middleware using the prototype and developing a conceptional framework containing guidelines and rules for the practitioner how to implement an enterprise system based on the adaptive middleware for near-time processing

References

[1] J. Fleck, “A distributed near real-time billing environment,” in Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA ’99, 1999, pp. 142–148.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch, Enterprise Application Integration: Grundlagen, Konzepte, Entwurfsmuster, Praxisbeispiele. Elsevier, Spektrum, Akad. Verl., 2006.

 ISBN: 978-1-61208-341-4

[3] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[4] Apache Camel. http://camel.apache.org. [retrieved: March 2014].

[5] D. Chappell, Enterprise Service Bus. Sebastopol, CA, USA: O'Reilly Media, Inc., 2004.

[6] S.-H. Chang, H. J. La, J. S. Bae, W. Y. Jeon, and S. D. Kim, "Design of a dynamic composition handler for esb-based services," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 287–294.

[7] X. Bai, J. Xie, B. Chen, and S. Xiao, "Dresr: Dynamic routing in enterprise service bus," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 528–531.

[8] B. Wu, S. Liu, and L. Wu, "Dynamic reliable service routing in enterprise service bus," in Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE, Dec 2008, pp. 349–354.

[9] G. Ziyaeva, E. Choi, and D. Min, "Content-based intelligent routing and message processing in enterprise service bus," in Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on, Aug 2008, pp. 245–249.

[10] A. Jongtaveesataporn and S. Takada, "Enhancing enterprise service bus capability for load balancing," W. Trans. on Comp., vol. 9, no. 3, Mar. 2010, pp. 299–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1852392.1852401

[11] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," Software Engineering, IEEE Transactions on, vol. 37, no. 3, May 2011, pp. 387–409.

[12] L. González and R. Ruggia, "Addressing qos issues in service based systems through an adaptive esb infrastructure," in Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, ser. MW4SOC '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/2093185.2093189

[13] R. Friedman and R. V. Renesse, "Packing messages as a tool for boosting the performance of total ordering protocls," in Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, ser. HPDC '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 233–.

[14] R. Friedman and E. Hadad, "Adaptive batching for replicated servers," in Reliable Distributed Systems, 2006. SRDS '06. 25th IEEE Symposium on, 2006, pp. 311–320.

[15] P. Romano and M. Leonetti, "Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning," in Computing, Networking and Communications (ICNC), 2012 International Conference on, Jan 2012, pp. 786–792.

[16] D. Didona, D. Carnevale, S. Galeani, and P. Romano, "An extremum seeking algorithm for message batching in total order protocols," in Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on, Sept 2012, pp. 89–98.

[17] "Amazon ec2 auto scaling," http://aws.amazon.com/autoscaling, [retrieved: March 2014].

[18] Auto scaling on the google cloud platform. https://cloud.google.com/developers/articles/auto-scaling-on-the-google-cloud-platform. [retrieved: March 2014].

 ISBN: 978-1-61208-341-4

Intermittently Updated Simplified Proportionate Affine Projection Algorithm

Felix Albu, Henri Coanda, Dinu Coltuc, Marius Rotaru
Dept. of Electronics
Valahia University of Targoviste
Targoviste, Romania
E-mails: {felix.albu, coanda, coltuc}@valahia.ro; marius.rotaru@gmail.com

***Abstract*—In this paper, an intermittent update interval for filter coefficients and a simplified output error vector computation is proposed for a proportionate affine projection algorithm. It is shown that the proposed algorithm has good convergence performance and much smaller computation complexity than other proportionate-type APAs. Also, the accuracy of its implementation using the logarithmic number system was investigated. We demonstrated the performance of the proposed algorithm for echo cancellation and adaptive feedback cancellation applications.**

Keywords-Proportionate-type algorithms; adaptive filters; affine projection algorithm; logarithmic number system.

I. INTRODUCTION

There are many adaptive algorithms proposed for adaptive systems [1][2]. The most used algorithms are: the Normalized Least Mean Square (NLMS) algorithm, the Affine Projection Algorithm (APA) [3], and fast versions of APA for various applications like echo cancellation, hearing aids and active noise control (e.g., [4]–[9]). It is known that in echo cancellation systems, the echo paths are often sparse [1]. An intuitive idea for this case is to exploit the sparseness of the echo path by updating filter coefficients independently and proportionally to their estimated magnitude. One of the first such algorithms was proposed by Duttweiler [10], and it was called the Proportionate Normalized Least-Mean-Square (PNLMS) algorithm. Several proportionate algorithms were designed (e.g., [11], μ-law PAPA [12], Improved PAPA (IPAPA) [13], Memory IPAPA (MIPAPA) [14], μ-law MIPAPA (MMIPAPA) [15], and Approximated MIPAPA (AMIPAPA) [16]). The latter algorithm is still too complex, and an approximation for the output error computation of AMIPAPA was proposed in [17]. It was termed Simplified AMIPAPA (SAMIPAPA) and the complexity reduction come at a price of a reduced performance by several dB, especially when using speech signals and sparse echo paths. In [18], an algorithm that uses a combination of recursive filtering, dichotomous coordinate descent iterations and an approximation of a matrix in order to further reduce its numerical complexity in terms of multiplications was also proposed.

Therefore, a new proportionate algorithm with little performance degradation that incorporates an approximation of the output error and an intermittent update of filter coefficients depending on a computed threshold [19][20] is proposed in this paper. The algorithm proposed by Albu et al. in [20] used an intermittent update on an affine projection algorithm. It is shown that the threshold derived for the affine projection algorithm by Shin, Sayed & Song in [21] it is good enough for the proposed proportionate APA. The new algorithm is termed Intermittently Updated SAMIPAPA (IUSAMIPAPA). IUSAMIPAPA distinguishes from the algorithm proposed by Albu et al. in [20], called Intermittently Updated APA (IU-APA), because it is a proportionate-type algorithm and uses other steady-state MSE estimation formula. Also, the update formula of [20] is related linearly to the logarithm of the estimated variance of the filter output error. IUSAMIPAPA is different from the algorithm proposed by Albu in [18] because it does not include DCD iterations and uses other approximation. The algorithm proposed in Albu and Kwan [22] is a sign algorithm without an intermittent weights update unlike the proportionate algorithm presented in this paper.

The paper is organized as follows. Section 2 presents a short overview of the proportionate-type algorithms for echo cancellation. In Section 3, SAMIPAPA is derived and the proposed intermittently updated SAMIPAPA is investigated. In Section 4, the proposed algorithm is compared with AMIPAPA and SAMIPAPA in the context of echo cancellation and adaptive feedback cancellation. Also, the accuracy of its simulation using the logarithmic number system is verified. Finally, the conclusions are given in Section 5.

II. PROPORTIONATE-TYPE ALGORITHMS

In an echo cancellation system, we consider the far-end signal $x(n)$, and the reference signal $d(n)$, where n is the time index. The adaptive FIR filter is given by the coefficients vector $\hat{\mathbf{h}}(n)=\left[\hat{h}_0(n),\hat{h}_1(n),...,\hat{h}_{L-1}(n)\right]^T$, where L is the length of the adaptive filter and superscript T denotes transposition. The error signal is given by [1]

$$e(n)=d(n)-\hat{\mathbf{h}}^T(n-1)\mathbf{x}(n) \tag{1}$$

where $\mathbf{x}(n) = [x(n), x(n–1),\ldots, x(n–L+1)]^T$ is a vector containing the L most recent samples of the input signal. If p is the projection order, the error signal vector is given by

 ISBN: 978-1-61208-341-4

$$\mathbf{e}(n)=\mathbf{d}(n)-\mathbf{X}^T(n)\hat{\mathbf{h}}(n-1), \quad (2)$$

where $\mathbf{X}(n)=\left[\mathbf{x}(n),\mathbf{x}(n-1),\ldots,\mathbf{x}(n-p+1)\right]$ is the input signal matrix, $\mathbf{d}(n)=\left[d(n),d(n-1),\ldots,d(n-p+1)\right]^T$ is the reference signal vector, and $\mathbf{e}(n)=\left[e(n),e(n-1),\ldots,e(n-p+1)\right]^T$ is the error vector.

The coefficients of the proportionate-type affine projection algorithms (PAPA) are updated as follows [18]

$$\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\mu\mathbf{G}(n-1)\mathbf{X}(n)\times \left[\delta\mathbf{I}_p+\mathbf{X}^T(n)\mathbf{G}(n-1)\mathbf{X}(n)\right]^{-1}\mathbf{e}(n), \quad (3)$$

where $\mathbf{G}(n-1)$ is an L x L diagonal matrix, δ is a regularization constant, μ is the normalized step-size parameter, and $\mathbf{I}_p$ is the p x p identity matrix. In the case of the improved PAPA (IPAPA) [13], the diagonal elements of $\mathbf{G}(n-1)$, denoted by $g_l(n-1)$, are evaluated as

$$g_l(n-1)=\frac{1-\alpha}{2L}+(1+\alpha)\frac{\left|\hat{h}_l(n-1)\right|}{2\sum_{i=0}^{L-1}\left|\hat{h}_i(n-1)\right|+\xi}, \quad (4)$$

where $-1\le\alpha<1$, $0\le l<L-1$ and ξ is a small positive constant. Let us denote [14]

$$\mathbf{P}(n)=\mathbf{G}(n-1)\mathbf{X}(n) =\left[\mathbf{g}(n-1)\odot\mathbf{x}(n)\ldots\mathbf{g}(n-1)\odot\mathbf{x}(n-p+1)\right], \quad (5)$$

where $\mathbf{g}(n-1)$ is a vector containing the diagonal elements of $\mathbf{G}(n-1)$ and the operator $\odot$ denotes the Hadamard product [14]. $\mathbf{P}(n)$ is approximated with

$$\mathbf{P}'(n)=\left[\mathbf{g}(n-1)\odot\mathbf{x}(n)\ldots\mathbf{g}(n-p)\odot\mathbf{x}(n-p+1)\right], \quad (6)$$

where $\mathbf{g}(n-k)$ are the vectors containing the diagonal elements of the matrixes $\mathbf{G}(n-k)$, with $k=1, 2, \ldots, p$ [14]. We have

$$\mathbf{P}'(n)=\left[\mathbf{g}(n-1)\odot\mathbf{x}(n)\quad \mathbf{P}'_{-1}(n-1)\right], \quad (7)$$

where the matrix

$$\mathbf{P}'_{-1}(n-1)= \left[\mathbf{g}(n-2)\odot\mathbf{x}(n-1)\ldots\mathbf{g}(n-p)\odot\mathbf{x}(n-p+1)\right], \quad (8)$$

contains the first $p-1$ columns of $\mathbf{P}'(n-1)$. The MIPAPA equations are written as in [16]:

$$\mathbf{S}_1(n)=\delta\mathbf{I}_p+\mathbf{X}^T(n)\mathbf{P}'(n) \quad (9)$$

$$\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\mu\mathbf{P}'(n)\mathbf{S}_1^{-1}(n)\mathbf{e}(n) \quad (10)$$

The coefficients of the approximated MIPAPA (AMIPAPA) are given by [16]

$$\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\mu\mathbf{P}'(n)\mathbf{S}_2^{-1}(n)\mathbf{e}(n) \quad (11)$$

where, $\mathbf{S}_2(n)$, is updated by changing both its first row and column with $\mathbf{X}^T(n)\mathbf{P}'_{:,1}(n)$ and adding δ to the first element. $\mathbf{P}'_{:,1}(n)$ denotes the first column of $\mathbf{P}'(n)$ and is given by $\mathbf{g}(n-1)\odot\mathbf{x}(n)$. The bottom-right $(p-1)\times(p-1)$ submatrix of $\mathbf{S}_2(n)$ is replaced with the top-left $(p-1)\times(p-1)$ submatrix of $\mathbf{S}_2(n-1)$ [16].

III. INTERMITTENTLY UPDATED SIMPLIFIED AMIPAPA

Firstly, an important numerical complexity reduction is obtained if $\mathbf{e}(n)=\mathbf{d}(n)-\mathbf{X}^T(n)\hat{\mathbf{h}}(n-1)$ is approximated as in the original fast affine projection algorithm [4]

$$\mathbf{e}(n)=\left[e(n);(1-\mu)\bar{\mathbf{e}}^T(n-1)\right]^T, \quad (12)$$

where $\bar{\mathbf{e}}(n-1)$ represents the first $p-1$ elements of $\mathbf{e}(n-1)$. The algorithm proposed in [17] used (12) instead of (2) and was called simplified AMIPAPA (SAMIPAPA).

The numerical complexity of the following algorithms in terms of multiplications is presented in equations (13)-(15) ($P_m=O(p^3)$ [23] indicates the numerical complexity in terms of multiplications):

$$C_{\text{MIPAPA}}=L(4p+1)+p+P_m \quad (13)$$

$$C_{\text{AMIPAPA}}=L(3p+2)+p+P_m \quad (14)$$

$$C_{\text{SAMIPAPA}}=L(2p+3)+2p+P_m. \quad (15)$$

It can be noticed that the complexity of SAMIPAPA is roughly half of that of MIPAPA for typical echo cancellation systems where $L\gg p$. However, the complexity can be further reduced using the intermittently updated procedure proposed in [19]. Thus, the update equation of (11) can be replaced by

$$\hat{\mathbf{h}}(n)=\begin{cases}\hat{\mathbf{h}}(n-1)+\mu\mathbf{P}'(n)\mathbf{S}_2^{-1}(n)\mathbf{e}(n), & \text{if } n \bmod i_n=0\\ \hat{\mathbf{h}}(n-1) & \text{otherwise}\end{cases} \tag{16}$$

where i_n is the computed update interval at time n. Starting with an initial update interval of 1, i_n is given by

$$i_n=\begin{cases}\max\left[1,i_{n-1}-1\right], & \text{if } e^2(n)\geq\gamma\\ \min\left[i_{n-1}+1,i_M\right] & \text{otherwise}\end{cases} \tag{17}$$

where i_M is the maximum update interval and γ is the threshold [19] computed as in (18)

$$\gamma=\frac{\mu\sigma_v^2 p}{2-\mu}+\sigma_v^2, \tag{18}$$

where σ_v^2 is estimated during silences [24]. The numerical savings are important because (11) requires $Lp+P_m$ multiplications and the filter can have hundreds of coefficients in echo cancellation systems. The update of the filter coefficients from (16) is performed only when $n \bmod i_n=0$ and not at every iteration like in (11). The new algorithm is termed Intermittently Updated SAMIPAPA (IUSAMIPAPA). The algorithm can have a periodic update if the update interval is fixed to $i_n>1$.

IV. SIMULATION RESULTS

Most of the simulations were performed in the context of echo cancellation, where the input signal is either white Gaussian noise or speech. The first impulse response from ITU-T G168 Recommendation [25] is padded with zeros in order to have 512 coefficients. A white Gaussian noise with a SNR = 30 dB is added at the output of the echo path. The performance measure used is the normalized misalignment (in dB), defined as $20\log_{10}(\|\mathbf{h}-\hat{\mathbf{h}}(n)\|_2/\|\mathbf{h}\|_2)$, where $\mathbf{h}$ denotes the true impulse response of the echo path. In the simulations with white noise, the performance curves are averaged over 10 independent trials. The regularization constant is $\delta = 0.01$, $p = 8$ and $\alpha = 0$. In all the simulations where the input signal is a white signal, the step size of all algorithms is 0.11.

Figure 1 shows the misalignment performance of the periodic SAMIPAPA with fixed periodically updated filter coefficients. It can be noticed that the larger the update interval, the lower steady-state error and the slower the convergence speed. Therefore, similar conclusions as those of [18] and [19] are obtained and this indicates that a variable updating interval for SAMIPAPA could lead to a good compromise between fast convergence and low steady-state error.

Figure 2 shows the misalignment curves for the proposed IUSAMIPAPA ($i_M=8$), SAMIPAPA, and the periodic SAMIPAPA with $i=8$. An abrupt change of the echo path after 25000 iterations by shifting the impulse response to the right by 12 samples was introduced in order to verify the tracking ability of the algorithms. It can be seen that IUSAMIPAPA has roughly the same initial convergence as SAMIPAPA and steady-state error of the periodic SAMIPAPA. The update of the filter weights is made on average only on a fifth of the number of iterations. Overall, for the investigated case, IUSAMIPAPA obtains an impressive 35% complexity reduction over SAMIPAPA in terms of multiplications (SAMIPAPA has 9884 multiplications, while IUSAMIPAPA has 6495 multiplications).

Figure 3 shows the misalignment curves for IUSAMIPAPA for different update intervals.

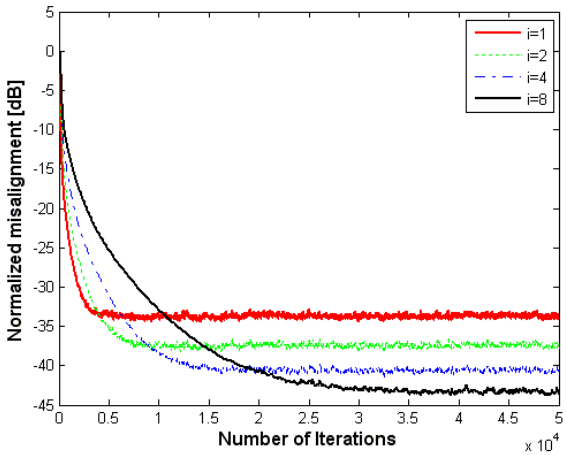


Figure 1. Misalignment of periodic SAMIPAPA for different update intervals, white noise, $p = 8$, $L = 512$, SNR = 30 dB.

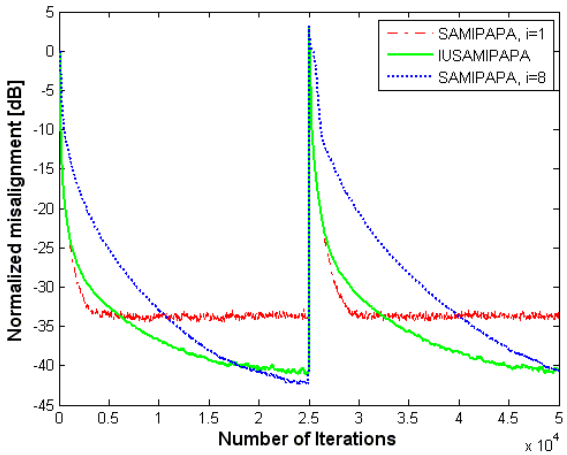


Figure 2. Misalignment of SAMIPAPA, periodic SAMIPAPA, $i = 8$, and IUSAMIPAPA $i_M=8$. Other conditions are the same as in Figure 1.

Similar conclusions with those obtained in [18] and [19] are obtained regarding the influence of i_M. It can be seen that the time to reach steady-state increases with i_M value.

For the considered case, the percentage of updates is about 15% for $i_M = 8$, 9% for $i_M = 16$, and 6% for $i_M = 32$. The overall number of updates is reduced by increasing i_M. The maximum update interval is set to the projection order in the following simulations. An example of computed i_n values and their histogram for the case $i_M = 8$ (Figure 3) is shown in Figure 4. It can be seen that during the initial convergence, the updating intervals are closer to 1, while they are closer to 8 in the steady-state region.

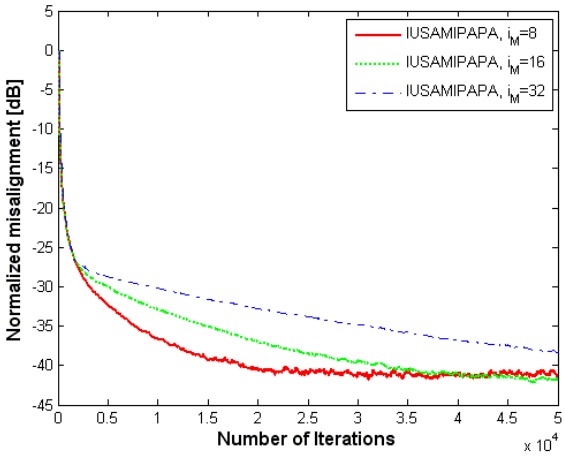


Figure 3. Misalignment of IUSAMIPAPA with $i_M = 8$, $i_M = 16$ and $i_M = 32$ respectively. Other conditions are the same as in Figure 1.

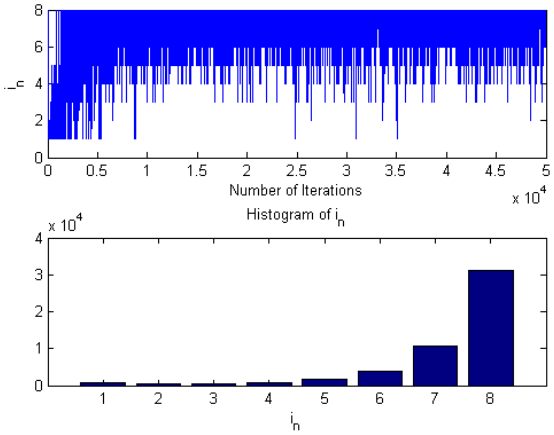


Figure 4. Computed update interval values (upper); and histogram of computed i_n values (lower)

In Figure 5, the input signal is speech, with p = 8, the output of the echo path is corrupted by independent white Gaussian noise SNR = 30 dB and the echo path changes after 0.5 seconds. The step-size for all algorithms is 0.2 for the following simulation. It was shown in [16] that MIPAPA has virtually identical performance with AMIPAPA at a higher computational cost. Therefore, for the following simulations, there is no need to plot the misalignment curves of MIPAPA. Also, the superiority of MIPAPA to APA for echo cancellation applications has been proved in previous publications [14]-[16]. Figure 5 shows that the approximation used by SAMIPAPA and the intermittent update of filter weights lead to slightly reduced performance (1 to 3 dB for this example) in comparison with AMIPAPA in case of a speech signal input. However, IUSAMIPAPA offers a better performance/complexity tradeoff than AMIPAPA, due to its reduced numerical complexity by about 42% (7766 multiplications vs. 13460 multiplications).

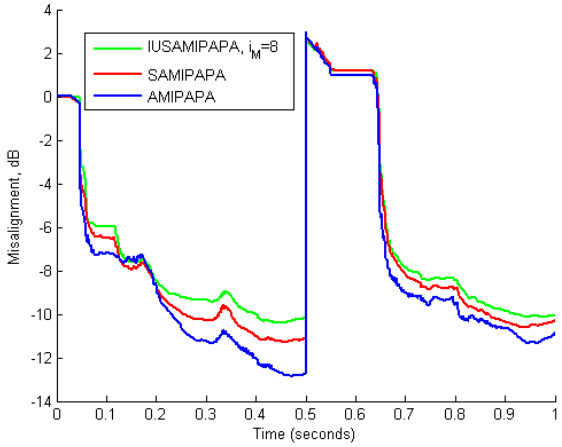


Figure 5. Misalignment of the AMIPAPA, SAMIPAPA and IUSAMIPAPA. Speech sequence, p = 8, L = 512, SNR = 30 dB, and echo path changes at time 0.5s.

The same conclusions can be drawn for results using colored noise as input signal, different filter lengths or maximum projection orders.

In the next simulation, the performance of MMIPAPA [15], AMIPAPA [16], SAMIPAPA [17] and IUSAMIPAPA is investigated in the acoustic feedback context [26]. The feedback path and the adaptive filter have 64 coefficients. A delay of 60 samples and a constant gain of 30 dB in the forward path were assumed. The sampling frequency was 16 kHz, $M = 8$, $\mu = 0.1$, and $\delta = 0.001$. The logarithmic factor of MMIPAPA [15] was 100. It can be seen from Figure 6, that most of the time, the performance of IUSAMIPAPA is superior to that of MMIPAPA, SAMIPAPA and AMIPAPA in case of a coloured input signal. IUSAMIPAPA obtains a smaller misalignment than the other algorithms, although has a slower convergence speed at some moments in time.

 ISBN: 978-1-61208-341-4

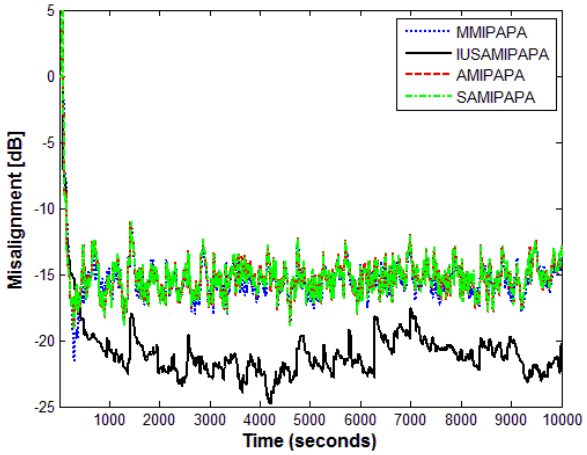


Figure 6. Misalignment of MMIPAPA, AMIPAPA, SAMIPAPA, and IUSAMIPAPA for an AFC application with coloured input signal, $M = 8$ and $\mu = 0.1$.

Figure 7 shows the same behaviour for a speech input signal. The parameters of the algorithms are the same as above example. It can be noticed that the performance of MMIPAPA, AMIPAPA and SAMIPAPA is most of the time similar. However, MMIPAPA has the highest numerical complexity from all the investigated algorithms. MMIP-APSA requires additional L logarithmic functions and L additions per iteration in comparison with MIPAPA.

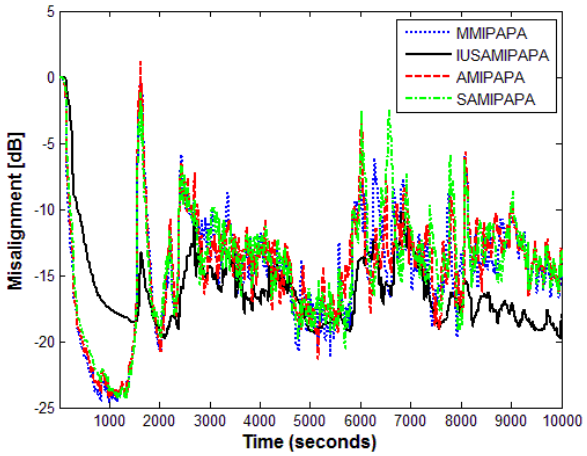


Figure 7. Misalignment of MMIPAPA, AMIPAPA, SAMIPAPA, and IUSAMIPAPA for an AFC application with speech input signal, $M = 8$ and $\mu = 0.1$.

We've also investigated the performance of the algorithm using 32-bit simulation using the logarithmic number system (LNS) and compared with 32-bit floating point results for the AFC example. The logarithmic number system is an alternative to floating-point that offers the potential to perform real multiplication, division and square-root at fixed-point speed and, in the case of multiply and divide, with no rounding error at all [27]. The logarithmic addition and subtraction are performed with the speed and accuracy equivalent to that of floating-point. The LNS format compares favorably against its floating-point counterpart, having greater range and slightly smaller representation error [27]. Impressive speed-ups were obtained over conventional floating point implementations for a wide range of algorithms [28][29]. More details about the logarithmic number system are available at http://www.ncl.ac.uk/eece/elm.

We considered the AFC experiment results for both 32-bit LNS and 32-bit floating point simulations. An accurate standard for comparison of the outputs was obtained by considering the corresponding double precision version results. The corresponding sum of absolute errors was computed for IUSAMIPAPA. The 32-bit LNS and 32-bit floating-point simulations have almost identical results. This confirmed similar conclusions obtained in the past for a wide range of algorithms. However, the sum of absolute errors of the 32 bit LNS implementation of IUSAMIPAPA was about 10% smaller than that of the 32-bit floating point implementation. Therefore, an LNS implementation could benefit from an increased accuracy.

V. CONCLUSION AND FUTURE WORK

In this paper, a low complexity proportionate-type AP algorithm was proposed. IUSAMIPAPA offers an excellent convergence performance/numerical complexity compromise in comparison with other proportionate AP algorithms. The performance was verified on an echo cancellation and adaptive feedback cancellation applications. Also, an accuracy investigation of an LNS implementation was performed. Future work will be focused on investigating the performance of the proposed algorithm on AFC application using two microphones in hearing devices [30] and compare it variable projection order versions [31].

ACKNOWLEDGMENT

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI project number PN-II-ID-PCE-2011-3-0097.

REFERENCES

[1] J. Benesty, T. Gaensler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, Advances in Network and Acoustic Echo Cancellation, Berlin, Germany: Springer-Verlag, 2001.

[2] E. Haensler and G. Schmidt, Eds., Topics in Acoustic Echo and Noise Control, Berlin, Germany: Springer-Verlag, 2006.

[3] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," Electron. Commun. Jpn., vol. 67-A, no. 5, May 1984, pp. 19–27.

 ISBN: 978-1-61208-341-4

[4] S. L. Gay and S. Tavathia, "The fast affine projection algorithm," Proc. of IEEE ICASSP, May 1995, pp. 3023-3026, doi: 10.1109/ICASSP.1995.479482.

[5] A. Gonzales, F. Albu, M. Ferrer, and M. Diego, "Evolutionary and variable step size affine projection algorithms for active noise control ", IET Signal Processing, vol. 7, (6), Aug. 2013, pp. 471-476, doi: 10.1049/iet-spr.2012.0213.

[6] S. Lee, I. Kim, and Y. Park, "Approximated affine projection algorithm for feedback cancellation in hearing aids," Computer Methods and Programs in Biomedicine, Vol. 87, (3), Sept. 2007, pp. 254-261, doi: 10.1016/j.cmpb.2007.05.014

[7] A. Gonzalez, M. Ferrer, F. Albu, and M. de Diego, "Affine projection algorithms: evolution to smart and fast multichannel algorithms and applications," Proc. of Eusipco 2012, Bucharest, Romania, Aug. 2012, pp. 1965-1969.

[8] F. Albu and A. Fagan, "The Gauss-Seidel pseudo affine projection algorithm and its application for echo cancellation," Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, 2003, Vol. 2, 9-12 Nov. 2003, pp. 1303 – 1306, doi: 10.1109/ACSSC.2003.1292199

[9] F. Albu and H.K. Kwan, "Combined echo and noise cancellation based on Gauss-Seidel pseudo affine projection algorithm," Proc. of IEEE ISCAS 2004, May 2004, pp. 505-508, doi:10.1109/ISCAS.2004.1328794.

[10] D. L. Duttweiler, "Proportionate normalized least-mean-squares adaptation in echo cancellers," IEEE Transactions on Speech and Audio Processing, vol. 8, no. 5, Sept. 2000, pp. 508–518.

[11] J. Benesty and S. L. Gay, "An improved PNLMS algorithm," in Proc. of IEEE ICASSP, 2002, vol. II, May 2002, pp. 1881–1884, doi: 10.1109/ICASSP.2002.5744994.

[12] H. Deng and M. Doroslovački, "Proportionate adaptive algorithms for network echo cancellation," IEEE Transactions on Signal Processing, vol. 54, no. 5, May 2006, pp. 1794–1803, doi: 10.1109/TSP.2006.872533.

[13] O. Hoshuyama, R. A. Goubran, and A. Sugiyama, "A generalized proportionate variable step-size algorithm for fast changing acoustic environments," Proc. of IEEE ICASSP, 2004, vol IV, May 2004, pp. 161-164, doi: 10.1109/ICASSP.2004.1326788.

[14] C. Paleologu, S. Ciochina, and J. Benesty, "An efficient proportionate affine projection algorithm for echo cancellation," IEEE Signal Processing Letters, vol. 17, no. 2, Feb. 2010, pp. 165–168, doi: 10.1109/LSP.2009.2035665.

[15] J. Yang and G.E. Sobelman, "Efficient μ -law improved proportionate affine projection algorithm for echo cancellation", Electronics Letters, vol. 47, Issue 2, Jan. 2010, pp. 73 – 74, doi: 10.1049/el.2010.7937.

[16] F. Albu, C. Paleologu, J. Benesty, and S. Ciochina, "A low complexity proportionate affine projection algorithm for echo cancellation," Proc. of EUSIPCO 2010, August 2010, pp. 6-10.

[17] F. Albu, "Simplified proportionate affine projection algorithm," Proc. of IWSSIP 2012, April 2012, pp. 382-385.

[18] F. Albu, "New proportionate affine projection algorithm," 41st International Congress and Exposition on Noise Control Engineering 2012, INTER-NOISE 2012, (9), Aug. 2012, pp. 7726 – 7733.

[19] K.-H. Kim, Y.-S. Choi, S.-E. Kim, and W. -J. Song, "An Affine Projection Algorithm with Periodically Evolved Update Interval," IEEE Trans on Circuits and Systems-II, vol. 58, no.11, Nov. 2011, pp. 763-767, doi: 10.1109/TCSII.2011.2168023.

[20] F. Albu, M. Rotaru, R. Arablouei, and K. Dogancay,"Intermittently-updated affine projection algorithm," Proc. of ICASSP 2013, May 2013, pp. 585 – 589, doi: 10.1109/ICASSP.2013.6637715.

[21] H.-C. Shin, A. H. Sayed and W.-J. Song, "Variable step-size NLMS and affine projection algorithms," IEEE Signal Processing Leters, vol. 11, no. 2, Feb. 2004, pp. 132–135, doi: 10.1109/LSP.2003.821722.

[22] F. Albu and H.K. Kwan, "New proportionate affine projection sign algorithms", in Proc. of ISCAS 2013, pp. 1789 – 1793, doi: 10.1109/ISCAS.2013.6571895.

[23] G. H. Golub and C. F. Van Loan, Matrix computation, 3rd edition. Baltimore, MD: The John Hopkins Univ. Press, 1996.

[24] J. Benesty, H. Rey, L. Rey Vega, and S. Tressens, "A nonparametric VSS NLMS algorithm," IEEE Signal Processing Letters, vol. 13, no. 10, Oct. 2006, pp. 581–584, doi: 10.1109/LSP.2006.876323.

[25] Digital Network Echo Cancellers, ITU-T Rec. G.168, 2002.

[26] M. Rotaru, C. Stanciu, S. Ciochina, F. Albu, and H. Coanda, "A FPGA Implementation of Prediction Error Method for Active Feedback Cancellation using Xilinx System Generator," Proc. of ADAPTIVE 2013, May 2013, pp. 26-29.

[27] J.N.Coleman, E.Chester, C.Softley and J.Kadlec "Arithmetic on the European Logarithmic Microprocessor," IEEE Trans. Comput. Special Edition on Computer Arithmetic, Vol. 49, No. 7, Jul. 2000, pp. 702-715, doi: 10.1109/12.863040.

[28] F. Albu, J. Kadlec, N. Coleman, and A. Fagan, "Pipelined Implementations of the A Priori Error-Feedback LSL Algorithm Using Logarithmic Arithmetic", Proceedings of ICASSP 2002, May 2002, pp. 2681-2684, doi: 10.1109/ICASSP.2002.5745200.

[29] F. Albu, J. Kadlec, C. Softley, and R. Matousek,, A. Hermanek, A. Fagan, N. Coleman, "Implementation of (Normalized) RLS Lattice on VIRTEX", Field Programmable Logic and Applications, Gordon Brebner and Roger Woods Editors, 2001, Aug. 2001, pp. 91-100, doi: 10.1007/3-540-44687-7_10.

[30] C. R. C. Nakagawa, S. Nordholm, F. Albu, W.-Y. Yan, "Closed-loop feedback cancellation utilizing two microphones and transform domain processing", ICASSP 2014, in press.

[31] F. Albu, C. Paleologu, and J. Benesty, "A Variable Step Size Evolutionary Affine Projection Algorithm," Proc. of ICASSP 2011, May 2011, pp. 429-432, doi: 10.1109/ICASSP.2011.5946432.

 ISBN: 978-1-61208-341-4

Application Independent Modeling and Simulation Environment for Systems with Self-aware and Self-expressive Capabilities

Tatiana Djaba Nya, Stephan C. Stilkerich
Airbus Group Innovations
Airbus Group GmbH
Ottobrunn, Germany
Email: {tatiana.djabanya, stephan.stilkerich}@eads.net

***Abstract*—Self-awareness and Self-expression in computer systems promise a lot of abilities enabling us to deal with the problems and challenges caused by their continuously increasing complex and heterogeneous structures and requirements, and the unpredictability and changes in their deployment environment. For this reason, engineering self-awareness and self-expression in computing systems has become a major research field in the computer science. To fill the gap between research at the conceptional level and the construction of first proof-of-concept demonstrators, a novel modeling and simulation environment for self-aware and self-expressive systems has been implemented. The environment is the Transaction-Level-Modeling (TLM) description in SystemC of the reference architectural framework for self-aware and self-expression systems. Therefore, it enables to simulate any topology of self-aware and self-expressive systems and deployed applications. This paper presents the said environment along with the developed reference architectural framework on which it is based, as well as an example motivated in the avionic domain.**

Keywords-SystemC; Transaction-Level-Modeling; Simulation; Self-awareness; Self-expression.

I. INTRODUCTION

Self-awareness and self-expression, which is adaptive behaviour based upon it, have proven to have a lot of benefits for computing systems [1]. A computing system with self-aware and self-expressive capabilities is for example able to deal with unpredictability and changes in its deployment environment; it is also possible to implement more functionality in such a system, such that it has the ability to execute the corresponding functions according to the knowledge it has of itself and its environment. Therefore, the engineering of self-awareness and self-expression in computing systems has become an emerging and major research field over the past years. In that regard, there are some very important issues or questions like: what are the requirements of a self-aware and self-expressive computing systems? How to properly engineer self-awareness and self-expression capabilities in a computing system? How to ensure the correctness of a system after self-adaptation operations? How to ensure and maintain the reliability, the fault tolerance level in a system after self-expression? [2][3][4].

To be able to address these questions and many others in this context, a reference architectural framework which structures the requirements of a self-aware and self-expressive system has been built. Based on this reference architecture, a modeling and simulation environment that can serve as support and test environment for the development and demonstration of developed concepts has been implemented. An alternative concept for realizing fault-tolerance in avionic systems using self-awareness and self-expression that has been developed has been used to validate the environment.

This paper presents the previous mentioned elements and is organized as follows: Section II describes the self-aware and self-expressive architectural framework. Section III presents the modeling and simulation environment. Section IV presents an example case of this environment representing a one single node avionic system and showing the alternative idea of fault tolerance for avionic systems. Finally, Sections V concludes the paper.

II. THE REFERENCE ARCHITECTURAL FRAMEWORK

Inspired from the biology and cognitive science, we defined in the EPiCS project [2] working definitions for self-awareness and self-expression in the context of a computing node [5][6]. Underlying these working definitions, we then developed the proposed reference architectural framework for a self-aware and self-expressive computing node [7]. This framework is shown here in Figure 1 and represents the conceptual components of a computing node with self-ware and self-expressive capabilities. Here, conceptual means that these components don't need to physically exit as separate components within an application, but provide a logical structure for reasoning about interactions between parts of a system, where these parts can have different levels of knowledge, autonomy and distributed decision making.

A. Self-Awareness

Self-awareness is achieved in this architecture by the sensors, the private and the public self-aware engines. As in agent architecture, the sensors are used here to collect information. Two types of sensors can be distinguished: first, the internal sensors named "Sensor" in the architecture, which collect information about the node internal state and second, the external sensors which collect information about the node's context. These are represented in the architecture by the conceptual components named "environment" and "Other nodes". Both, the private and public self-aware engines are responsible here for collection of information from the corresponding type of sensors and for the processing of this information. Moreover they will trigger the node's reaction and adaptation process after the information processing, if necessary. According to the working definition, a self-aware system may possess historical knowledge, predictors of future likely states or contextual

 ISBN: 978-1-61208-341-4

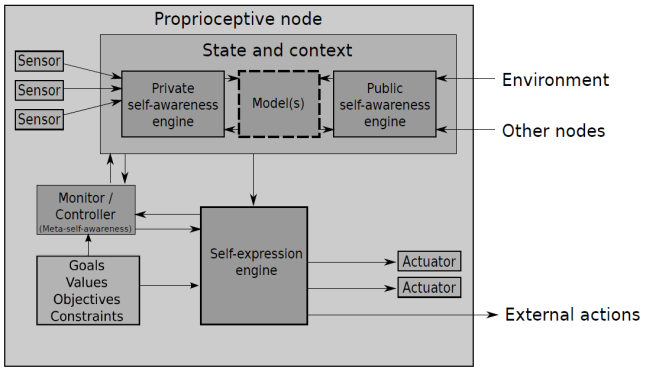


Figure 1. The conceptual components of a self-aware and self-expressive node.

information, in addition to purely instantaneous sensor readings. To enable this richer form of self-awareness, the self-awareness engines may engage in learning or modeling of information. We therefore introduce the possibility of internal models (online learning schemes), which are located in the component called ”Model(s)” and may be introduced as and when required in enabling the required level of self-awareness.

B. *Self-Expression*

To achieve self-expression, our architecture includes actuators and a self-expressive engine. The self-expressive engine has the role of taking decisions about the actions that must be performed by the node itself in order to adapt its behaviour. As required by the working definition, this decision making process always take into account the node’s state, context, goals, values, objectives and constraints which are available in the node and later in this engine through the conceptual component bearing the same name. The actions determined by the self-expressive engine are passed to the actuators which execute them. Actuators are represented in this architecture by the conceptual components called ”Actuator” and the ones called ”External actions”. As its name suggests, the latter execute the actions targeting the node’s environment. The ”actuators” for their part executed the chosen actions targeting the node itself.

C. *Meta-Self-Awareness*

Meta-Self-awareness is the higher level of self-awareness in a computing node and represents the ability of the node to be aware of its own awareness and to choose the level of awareness suitable to the node situation, to better achieve its goals. For this purpose, there is on the one hand a conceptual component named ”Monitor/Controller” in the reference architecture of the node. As shown in Figure 2 through the arrows, this component has access to the node’s goals, values, objectives and constraints, to the self-aware and self-expressive engines. In this way, it has a high-level view over the node’s behaviour and can intervene, when necessary, to lessen or increase the level of self-awareness and self-expression in the node.

III. The Modelling and Simulation Environment

The modelling and simulation environment has been implemented in SystemC at the transactional level, i.e., Transaction-Level Modelling (TLM). TLM is a modelling methodology which is primary concerned with the efficient modelling of bus systems and their transactions (hence the name TLM). It reaches a higher abstraction level over the register transfer level

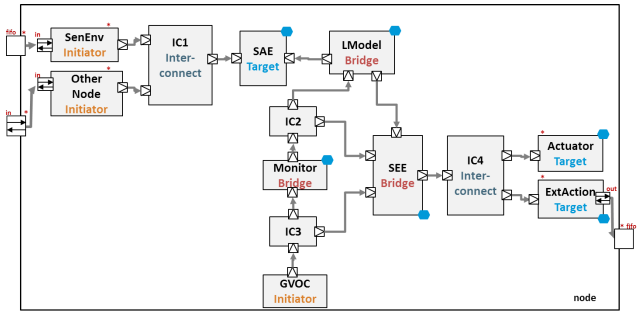


Figure 2. SystemC-TLM graphical view of the environment.

modelling and thus enables to implement virtual prototypes of systems which can be used to test developed software or to assess the performance of different system architectures through simulation. SystemC is a system description language. It enables both software and hardware description. The above mentioned properties and advantages of SystemC and TLM are the reasons which lead our decision to choose them for the realization of our modeling and simulation environment. The implemented SystemC TLM model is described in the following subsections. The next subsection gives an brief introduction in SystemC TLM in order to facilitate the understanding of the subsequent subsections focussed on the detailed proper description of the environment.

A. *Theoretical background*

A SystemC TLM Model is mainly composed of components which communicate among each other over sockets by initiating transactions by the means of processes.

A component has a role which can be of three types: initiator, target and interconnect. An initiator is able to initiate transactions to communication with other components; a target cannot initiate transactions and is always the target of a transaction. As for the interconnect component, it functions as a bus or a router for the transactions and usually execute address mapping operations. A component can act as an initiator for some transactions and as a target for others. In this case it is called a bridge.

As mentioned above, the communication in a SystemC TLM model occurs here in form of transactions (method calls) through which, in its simplest form, an initiator has the possibility to write or read data to/from its target component. The details of the transactions such as the size, the address and the type of data are regulated by the initiator when initiating the transactions and later by the interconnect components, if present, to ensure the correct routing of data.

To be able to initiate transactions, initiators need thread processes. A process describes the functional behaviour of a TLM component. The SystemC simulator implements a cooperative multitasking environment, i.e. some process instances execute without interruption, only a single process instance can be running at any time, and no other process instance can execute until the currently executing process instance has yielded control to the kernel. A process shall not pre-empt or interrupt the execution of another process. Each process has a sensitivity list which is a set of events and time-outs which can be defined during implementation to determine when it is executed or resumed by the scheduler.

B. *Model description*

The objective of achieving the functionality described in the reference architecture (Section II) of a self-aware and self-

 ISBN: 978-1-61208-341-4

TABLE I: DESCRIPTION OF THE TLM MODEL

Component	Role	Equivalent in the architecture	Transactions target(s)	Process(es)	Execution order	Transactions type
SenEnv	Initiator	Sensor, Environment	SAE	B	3	WRITE
OtherNode	Initiator	OtherNode	SAE	B	4	WRITE
GVOC	Initiator	Goals - Values - Objectives - Constraints	Monitor SEE	A1 A2	1 2	WRITE
LModel	Bridge	Model(s)	SAE SEE	C1 C2	5 7	READ WRITE
SEE	Bridge	Self-expressive Engine	Actuator ExtAction	D	9	WRITE
Monitor	Bridge	Monitor/Controller	LModel SEE	E1 E2	6 8	READ
SAE	Target	Private and Public Self-aware Engines				
Actuator	Target	Actuator				
Extaction	Target	External actions				
IC1, IC2, IC3, IC4	Interconnect					
node	Module			F		

expressive node as well as the data flow among its components in compliance with the rules and mechanisms of TLM lead us to the model presented in Figure 2. Complementary to Figure 2, Table I clearly listed the TLM components of this model, their roles, their processes, the transaction types and most importantly their equivalent in the reference architecture and the execution chronology of processes or rather transactions among them.

1) Functional Behaviour of components:
The first component to come into operation inside a node when the simulation is started is the Goals-Values-Objectives-Constraints (GVOC) component. As shown in Table I, it embodies the node's goals, values, objectives and constraints. It is an initiator and as such, it forwards the goals, values, objectives data respectively to the monitor through process A1 and to the component SEE through process A2. For this purpose, each of both processes A1 and A2 initiates WRITE transactions to the corresponding targets. The interconnect components IC3 placed between the GVOC component and its targets ensures the data sent by the GVOC components always reach the intended target.

The second component(s) to come into play are the sensors: the SenEnv first, then the OtherNodes components. They are respectively responsible for the gathering of private and public information inside the node. A node can possess as many sensors as necessary. Each of them has a process B that initiates WRITE transactions to forward its collected information to the common target component named SAE. The SAE acts as a memory on which every sensor possesses a reserved space for its data with read access only. In other words, the memory space on the SAE is divided equally between all the sensors components. The interconnect component IC1 placed between the sensors and the SAE in the figure ensures the correct addressing of the memories areas by the different sensors during transactions as well as the prevention from overwriting of data by a sensor in its reserved memory area. This is achieved by the implementation through a linear mapping function for transactions addresses.

The LModel component is the third initiator component to come into play after the simulation starts. It is responsible for the evaluation of sensor data available inside the node, i.e, in the SAE memory, on the one hand. On the other hand, it is responsible for the initiation of the self-expressive behavior of the node. Thus, its has a process C1 which initiates read transactions to read the sensor data out of the SAE memory and let them be evaluated. Through its second process C2, the LModel component finally initiates WRITE transactions to forward the results or the necessary information to the SEE component to trigger the self-expression of the node, if necessary.

After the LModel follows the SEE component. As described in the Table I, the SEE component is a bridge component which embodies the self-expressive engine. Accordingly, it analyses the information previously received from the LModel component and selects the action to be taken. Following this, its process D starts WRITE transactions either to the actuators embodied here by the target component of the Model bearing the same name or to the external actuators embodied here by the target components named ExtActions or to both.

The monitor, which embodies the Monitor/Controller of the self-aware and self-expressive node, is here implemented in its simplest form, which is a monitor of the self-aware and self-expressive engines' actions. To this end, it has a process E1 which initiates READ transactions towards the LModel component to read the report data of its actions stored in its internal report memory. This occurs immediately after the evaluation of the sensor data in C1. It also has a process E2 which, similarly to E1, initiates WRITE transactions to read the report data in the report memory of the SEE component right after the actions in the node has been taken. In contrast to other components, the monitor actions must not be executed in every transaction cycle. According to its needs, the user has the possibility to give the period for the activation of the monitor actions.

After the SEE component or the monitor is activated, the LModel component operates again. It reads the next available data out of the SAE memory and the whole process described above from that point on is repeated. This occurs until all the sensor data stored on the SAE memory are evaluated. Then, the whole operation cycle, named here as **process-cycle**, starts again. Here, the number of process-cycles inside a node during the simulation of a model depends on the whole amount of sensor information to be processed inside the node and in each process-cycle. The latter is defined by the user before simulation starts.

All the above described TLM components of a self-aware and self-expressive node are encapsulated inside a SystemC module named "node" as shown in Figure 2. This module represents the highest hierarchy in the model and is responsible

 ISBN: 978-1-61208-341-4

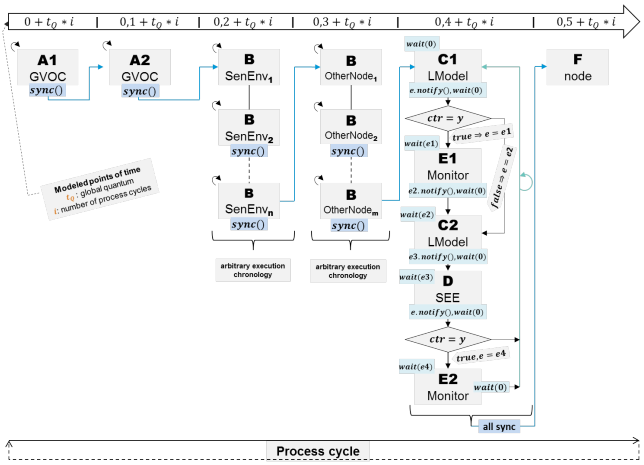


Figure 3. The implemented execution chronology of processes.

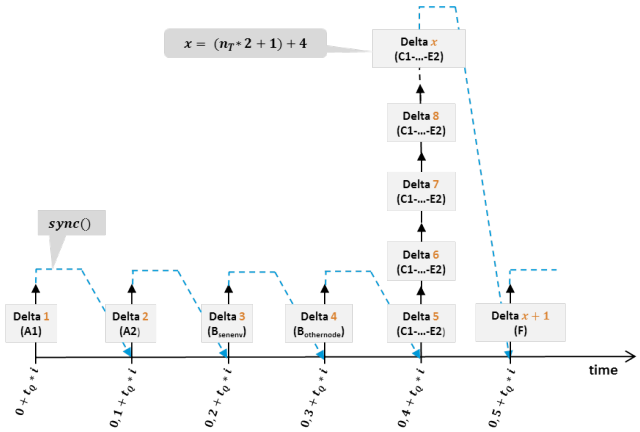


Figure 4. Delta-cycles within each process-cycle.

for the generation and instantiation of the TLM components inside each node according to the user specifications as well as for the resulting sockets and port-bindings for the communication among the components inside the node and between the system nodes at simulation start. Furthermore it has a process F that is the last to be executed in the process-cycle. Process F is just a synchronization process, i.e., it does not initiate any transactions, it is executed only once in each a process-cycle and ensures that processes in all node of a multi-node self-aware and self-expressive system end at the same simulation time in a process-cycle.

2) Processes:

From the model description above, it appears that there is a precise execution chronology of transactions and thus of processes in the environment that is a prerequisite and must always be maintain during simulation in order to reflect the functionality of a self-aware and self-expressive node prescribed by the architecture. This is: $\mathrm{A1} \rightarrow \mathrm{A2} \rightarrow \mathrm{B} \rightarrow \{\mathrm{C1} \rightarrow [\mathrm{E1}] \rightarrow \mathrm{C2} \rightarrow \mathrm{D} \rightarrow [\mathrm{E2}]\}$. It represents the so-called process-cycle previously mentioned. The processes within the curly brackets, here referred to as process chain, are executed alternately after each transaction until all sensor information available in the SAE memory are evaluated. Finally, the processes E1 and E2 of the monitor in the square bracket are activated in specific process-cycle intervals. A more detailed and precise view of the execution chronology of processes within a node is shown in Figure 3 and the following explain how this has been ensured through implementation.

For the temporal processes in a TLM Model, SystemC offers two possibilities [8]: The loosely-timed modeling style, which just models the start and end times of transactions but enables fast simulation times. The second one is the approximately-timed modeling style, which really details the phases of a transaction but at the cost of simulation performance. Giving the fact that the environment has to deal with industrial size systems, the simulation performance was a main concern during the implementation and has therefore lead us to the choice of loosely-timed modeling style. The Loosely timed modeling style implies the temporal decoupling of processes. This is implemented by means of the so called global time quantum t_{globQ}. The global time quantum is a time values that defines the synchronization (suspension) times of all processes t_{sync}. Each process run ahead simulation time and is executed as many times as possible till the next synchronization point is reached. The next synchronization point $t_{sync,next}$ of a process depends on the latency times of the transactions it executes after the previous synchronization and the actual simulation time t_{sim}. Within each process, the latency time t_{trans_delay} of each executed transaction is added up to the so-called local time offset t_{off}, which is then used to verify, if the process has to synchronize, i.e., be suspended. A suspended process can run again, only when the scheduler has advanced the simulation time t_{sim} of this same local time offset. So, for every process of the model the following formulas always hold:

$$\text{synchronization points:} \quad t_{sync} = N * t_{globQ}, \quad N \in \mathbb{N} \quad (1)$$

$$\text{between 2 subsequent } t_{sync}: \sum_{i=1}^{n} t_{trans\_delay,i} = t_{off}, n \in \mathbb{N}^* \quad (2)$$

$$\text{synchronization condition:} \quad t_{off} \geq t_{sync,next} - t_{sim} \quad (3)$$

From the above described behavior of temporal decoupled processes and their formulas, it results that the execution time of a process after a synchronization relies on the following three key parameters, which can be modified during implementation: The first execution time at simulation start, the global quantum and the local time offset, which is the sum of the latency times of the transactions.

In order to fix this execution and obtain the precise execution chronology illustrated in Figure 3, we, therefore firstly chose the adequate first execution times of some processes. Secondly, we determined the value range for the global quantum. Third and finally, with the assumption that all transactions of a process always have the same latency time, we decided to let the user input the number of transactions to be executed per process-cycles and we established formulas for the automatic calculation of the latency times of the transactions between the synchronization points during the elaboration and the generation of the simulation model, such that the local time offset between two synchronization points always equals the global quantum.

- The first executions times t_{beg}:

To fix the first execution times of processes in our environment, we make use of time-outs. A time-out occurs when the method $wait(t)$ is called with a time-object $t \in \mathbb{Z}^+$ as parameter. When executed, the running process is suspended and resumed after the given time period has elapsed.

So, with respect to the prerequisite execution chronology of processes and independently of the time unit, we respectively chose the following times t_{beg} for the processes A2, C1 and F: $0.1,\ 0.2, 0.5$. For the processes B's, for a better overview,

we chose $t_{beg} = 0.2$ respectively chose $t_{beg} = 0.3$ for the ones located inside the SenEnv components and $t_{beg} = 0.4$ for the others located in the OtherNodes components. This method call is executed only once at simulation start in each of these processes.

To achieve the alternate execution of processes in the process chain, we make use of time-outs and events. Each process of the chain notifies an event belonging to the dynamic sensitivity list of the next process and calls the method $wait()$ with $t = 0$ as parameter, after it has executed a transaction (see Figure 3). This produces a so-called delta-cycle, i.e., a process is suspended and resumed at the same simulation time, but in the next delta-cycle. So, the processing of a sensor data set and the reaction based upon the processing results always happens at the same simulation time but in different delta-cycles. And all sensor data set stored in a the SAE in a process-cycle are all processed by the process chain within the same process-cycle.

- The global time quantum t_{globQ}:

In [9], it is proved that the global time quantum of a TLM Model should be determined, in accordance with the whole simulation time period, so that the number of resulting synchronizations n_{sync} or delta cycles n_{delta_cycles} doesn't exceed a few hundred thousands. So the following inequalities should hold:

$$n_{sync} \leq 100000 \tag{4}$$

$$\text{or} \quad n_{delta\_cycles} \leq 100000 \tag{5}$$

Assuming that t_{sim} denotes the whole simulation period, the number of synchronizations in the model can be calculated with the following formula:

$$n_{sync} = \left\lfloor \frac{t_{sim}}{t_{globQ}} \right\rfloor \tag{6}$$

In our model, the number of generated delta-cycles by the processes between two synchronization points is always the same and is illustrated in Figure 4. The temporal decoupled processes A1, A2, B's, and F always generate one delta-cycle. Because of the additional time-outs used in process chain to ensure the prerequisite alternate behavior, as described in the previous paragraphs, the process chain always produces two delta-cycles to complete the evaluation of a single sensor data set. Given that the process chain has to evaluate all sensor data set available on the SAE memory within a process-cycle, the number of generated delta cycles by the process chain therefore depends on the number of (read) transactions initiated by process C1 during a process-cycle. Finally there is an additional delta-cycle generated at the end of the process chain's execution for the synchronization of its processes. Thus, we have:

$$n_{delta\_cycles} = (N_T * 2 + 6) * n_{sync} * n_{nodes\_nr} \tag{7}$$

where n_{nodes_nr} is the number of nodes in the simulated system and N_T is the number of (read) transactions of C1 between two subsequent synchronizations points. The formulas (4), (5), (6), (7) above lead us to the following formulas for the value range of the global time quantum:

$$\left\lfloor \frac{t_{sim}}{t_{globQ}} \right\rfloor \leq \frac{100000}{(N_T * 2 + 6) * n_{sync} * n_{nodes\_nr}} \tag{8}$$

- The latency times of transactions:

From the given synchronization condition (3) for temporally decoupled processes, it results that the global quantum is always less or equal to the sum of the latency times of all executed transactions between two synchronization points. Thus, with t_{trans_delay} denoting the latency time of the ith transaction of a process between two synchronizations points, we have:

$$t_{globQ} \leq \sum_{i=1}^{n} t_{trans\_delay,i} = t_{off} \tag{9}$$

Assuming that the value of local time offset is equal to the global quantum and that all the transactions between the synchronization points have the same latency times t_{trans_delay}, we were able to derive the formulas below for the number of transactions of the processes between every two subsequent synchronization points:

$$t_{trans\_delay} = t_{globQ}/N_T \tag{10}$$

where N_T denotes the number of transactions of each of the processes per process-cycle . This is given by the user before simulation start for the processes A1, A2 and B. Given the fact that process C1 and C2 have to read and evaluate all sensor information stored inside the node in a simulation cycle within the same simulation cycle, the number of transactions that they generate in each simulation cycle is equal to the mathematical product of the number of transactions n_{T_B} generated by each sensor and the number of available sensors n_s in the system. Thus,

$$N_{T_{C1,C2}} = n_{T_B} * n_s \tag{11}$$

This also applies the processes E1, E2 and D of the process chain, because they run in each simulation cycle as many times as the processes C1 and C2. Thus,

$$N_{T_{E1,E2,D}} = N_{T_{C1,C2}} \tag{12}$$

A transaction can be either a single transaction or a burst and the latency times of a single transaction t_{single_delay} differs from the latency times of a burst transaction, which is actually what formula (10) computes. The interrelation between both latency times is:

$$t_{trans\_delay} = t_{single\_delay} * BL \tag{13}$$

$$\text{with } BL = \left\lceil \frac{DL_{max}}{BUSWIDTH/8} \right\rceil$$

By substituting (11) in (10), we finally obtain the following formulas for the latency times of the single transactions for the processes A1, A2, B's

$$t_{single\_delay} = t_{globQ}/(n_{T_B} * BL) \tag{14}$$

and for the process chain:

$$t_{single\_delay} = t_{globQ}/(n_{T_B} * n_s * BL) \tag{15}$$

Applying the above computed formulas as well as the simulation-execution mechanisms as described above enabled us to meet our objective relative to the execution chronology of processes during simulation. For each process, we

$$t_i = t_{beg} + i * t_{globQ} \text{ mit } i = n - 1 \text{ und } i \in \mathbb{N} \tag{16}$$

where i denotes the i-*th* process-cycle in the simulation.

IV. USE CASE

For test and validation purposes, an avionic subsystem consisting of a single self-aware and self-expressive node with an alternative concept of fault-tolerance has been modeled and simulated using the introduced simulation and modeling environment. This concrete system is presented in the next section. Additionally the simulation results are presented and discussed.

A. Scenario description

The system under investigation is an avionic subsystem on which an application composed of a safety relevant thread C_r and two optional, i.e., not safety relevant, threads $O1$ and $O2$ is installed. The safety relevant thread is designed with triple modular redundancy [10] according to the reliability requirement standards [11]. This subsystem consists of a single self-aware and self-expressive node and the idea here is to make use of the self-aware and self-expressive capabilities of this node to drive fault tolerance and mitigation strategies.

In details, some physical properties of the system, here in our exemplary use case scenario the temperature, are measured by the sensors during service and compared with their known empirical values. Differently than in today's traditional fault tolerance designs, The 1st and 2nd copy of the critical thread, $C_r(1)$ and $C_r(2)$, are running at the system start. And its 3rd copy, $C_r(3)$ is only generated and turned on in case of discrepancy between the measured temperature values and the given empirical values of the temperature. At the same time, the optional threads are progressively shut down. Both, the generation and the turn-on procedure of the third copy of C_r as well as the turn-off procedures of the optional threads happen progressively. The objective here is to secure the operation of the critical thread C_r, which execute the safety relevant operations of the subsystems. If the measured value of the temperature still hasn't fall back in the desired value range after the actions cited above have been taken, then the system is restarted and the threads are bring back to the start configuration. But, as soon as the temperature values measured by the sensors comply with the given empirical values, the optional threads are progressively turned on while the 3rd copy of the critical thread is progressively switched back and deleted.

An example of the modification of the threads' execution state according to the monitoring of the system temperature as described above is illustrated in Figure 5.

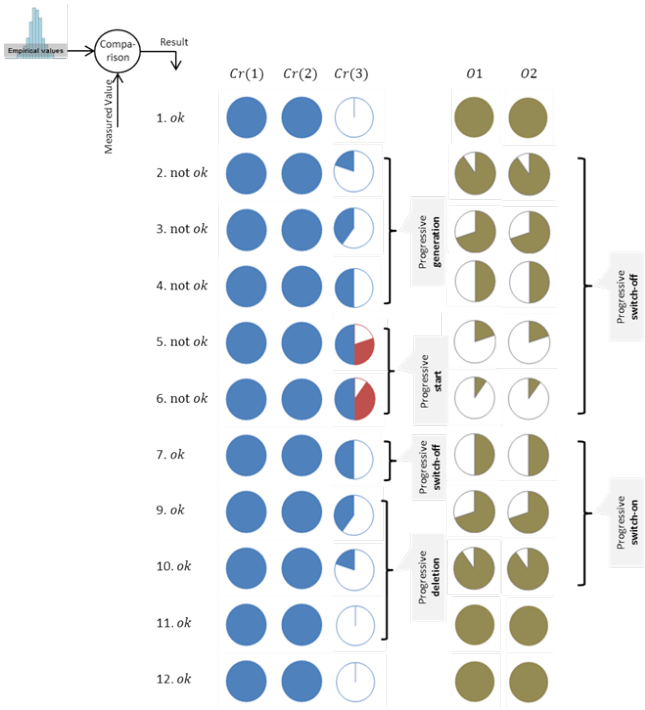


Figure 5. Example of the threads' execution state in the system according to the comparison results.

B. Prototyp Building

1) The threads:

To model this concrete system, we implement the threads as classes with a constructor parameter of type *string* representing the type of the thread, here $typ = \{critical, optional\}$ and a value s of type *float* representing the state of a thread, here $s \in [0, 1]$. During simulation, with respect to the results of the continuously comparison between the measured and empirical value of the temperature (self-awareness), this state variable is altered by the actuator of the node (self-expressive behavior) to model the behavior of the corresponding threads. For the optional threads, this indicates the progressive switch-on and off processes. For the critical thread Cr(3), it indicates its progressive generation, switch-on, switch-off as well as its deletion. For an optional thread, i.e, $typ = critical$, $s = 0$ means that the thread is switched off and $s = 1$ means that the thread is switched on or running. For a critical thread, $s = 0$ means that it is deleted, $s = 0.5$ means that it is generated and $s = 1$ means that it is switched on or running.

2) The functionality of the components:

- The SenEnv component

Due to the fact that, there is only one physical parameter to be observed, the system's own temperature, the model only needs one SenEnv component. The measured temperature values are given here in a matlab file. Thus, the SenEnv component reads out the file at program start and stores the data in a vector. In each process-cycle, the SenEnv component executes a transaction to transfer a single temperature value t of the vector to the SAE component.

- The LModel component

The value previously stored in the SAE component by the sensor is read by the LModel and compared with the given maximal empirical value T_{max} of the system temperature. Using the given frequency distribution of the temperature empirical values and depending on the comparison results, a value $v(t)$, which will help to initiate the self-expressive behavior of the system, is computed as follows:

$$v(t) = \begin{cases} -1.0 & \text{when } t > T_{max} \\ N[i] & \text{else} \end{cases} \quad \text{with } i = round(t - T_{max}) \tag{17}$$

The frequency distribution is given as a matlab file and stored at simulation start in the LModel component in a vector N. Here, $N[i]$ denotes the frequency density of this distribution. The computed value of $v(t)$ is thereupon transferred to the SEE component by the process C2 of LModel.

- The SEE component

This component uses the received value v(t) to compute a so-called decision vector $E(v)$. This vector consists of 5 values and represents the choice of the self-expressive engine of the node regarding the action to be performed on the threads

according to the node's awareness and reasoning.

$$E(v) = (\underbrace{e_1}_{Cr(1)} \quad \underbrace{e_2}_{Cr(2)} \quad \underbrace{e_3}_{Cr(3)} \quad \underbrace{e_4}_{O1} \quad \underbrace{e_5}_{O2}) \tag{18}$$

with $e_i \in [-1, 1]$and $i \in \{1, 2, 3, 4, 5\}$. So, each element of this vector E is the update value to be used by the actuator to alter the state of the corresponding thread of the system. In case that the system must restart, the SEE component will compute the following decision vector: $E(v) = \{-1, -1, -1, -1, -1\}$, else the following formula is used:

$$\begin{aligned} e_i(v) &= \begin{cases} -0.01 & \text{with } v = -1 \\ v - N_{max} & \text{with } v \leq 0.9 * N_{max} \\ v - N_{max} + 0.1 & \text{else} \end{cases} \\ e_3(v) &= -1 * e_i(v) \end{aligned} \tag{19}$$

with $i \in \{4, 5\}$ and N_{max} maximal value of the frequency density. After computation, the SEE component finally forwards this vector to the actuator of the node.

- The actuator component

As already mentioned above, the actuator component uses the received decision vector $E(v)$ to update the threads execution state according to the nodes self-awareness. So, for each of the threads i we have the following:

$$s_{new,i} = s_{actual,i} + e_i. \tag{20}$$

During the simulation, the computed values in each process-cycle are stored in a matlab file.

- The monitor component

Here, the task of the monitor is to read the report data of the SAE and SEE components and to monitor them, i.e., to write them in a matlab file that is used to control the values $v(t)$ and $E(v)$ computed respectively by the SAE and the SEE components.

- The GVOC component

The only constraint given here is the maximum value T_{max} for the system temperature. So, in each process-cycle, GVOC just forwards T_{max} to the SEE and Monitor components.

- The OtherNode and the ExtActions components

As we have mentioned above, this system under investigation comprises a single node. Thus, this prototype don't need any OtherNode component. In addition, there is no actions to be performed on the node's environment. This implies that there is also no need for ExtActions components in this prototype.

C. Simulation and Evaluation

With a total of 2500 given temperature values to be processed inside the self-aware and self-expressive node, the number of temperature value to be process within a process-cycle set to one, i.e., one transaction of the sensor per process-cycle, a simulation time period $t_{sim} = 5002ms$ and a global quantum time of $2ms$ were chosen according to (8) derived in section 3. Our model needed exactly 17500 transactions for all processes, 2001 synchronisations and lasted around 3 minutes. A total of 16008 delta-cycles were generated, which agrees with (7).

Figure 6 displays on the top line chart, the run of the given measured temperature values and, on the bottom line chart, the run of the threads states values computed over the whole simulation period according to the temperature values. Here, the maximal empirical temperature value is $T_{max} = 87\,^{\circ}\mathrm{C}$. One can realize from this illustration that the temperature of

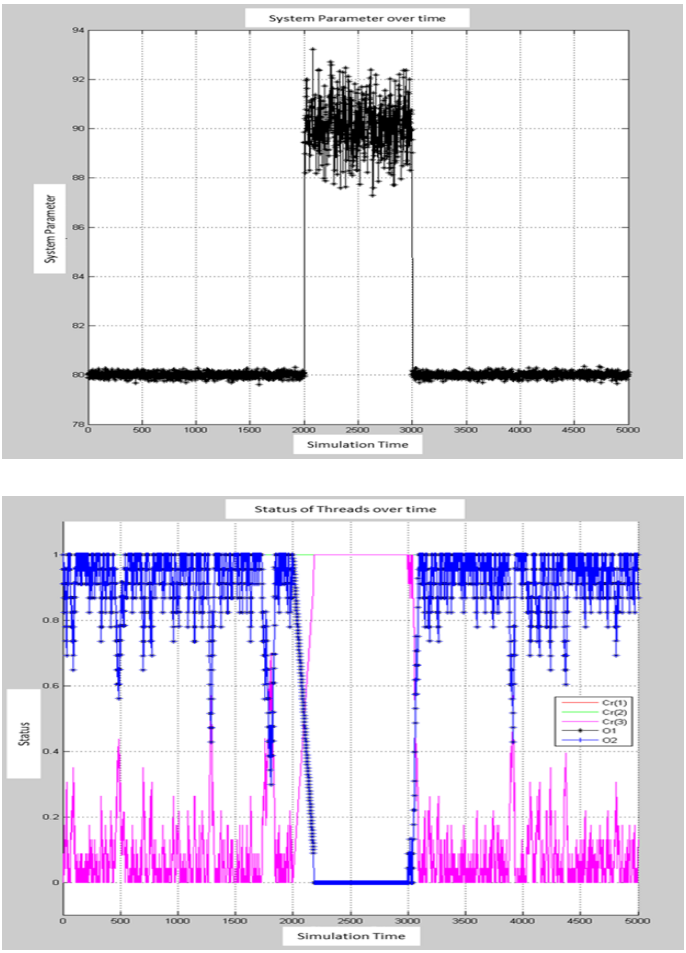


Figure 6. Temperature and threads execution state during the simulation period.

the system is not constant. But the first and last thousand values remain below the given maximum T_{max} while the remaining five hundred values exceed it. As expected, the state values s of the threads vary accordingly over the simulation period. Indeed, the state value of $Cr(3)$, the 3rd copy of the critical thread, is also not constant but always remains under the value 0.5 when the temperature isn't to near to the given maximum. When the temperature continues to rise, approaches, reaches or exceeds the given maximum, $Cr(3)$ is generated ($s_{Cr(3)} = 0.5$), progressively switched on and remains in this state ($s_{Cr(3)} = 1$). Meanwhile, the optional threads are switched off ($s_{O1,O2} = 0$). As soon as the temperature falls back, the state value of $s_{Cr(3)}$ decreases while s_{O1} and s_{O2} increase. Some specific points of the simulation have been captured in Figure 7 and underpin the above statement.

V. Conclusion

In this paper, we described a reference architectural framework developed to structure the requirements for the design of computing system with self-aware and self-expressive behaviour. Subsequently, we presented a modelling and simulation environment developed in SystemC using the Transaction-Level Modelling (TLM) and based on the previous mentioned framework for the construction of prototype and the tests and validation of novel concepts developed based on both properties. The presented environment can be used to build virtual prototypes of self-aware and self-expressive systems for industrial systems. Moreover, through the clear separation between

 ISBN: 978-1-61208-341-4

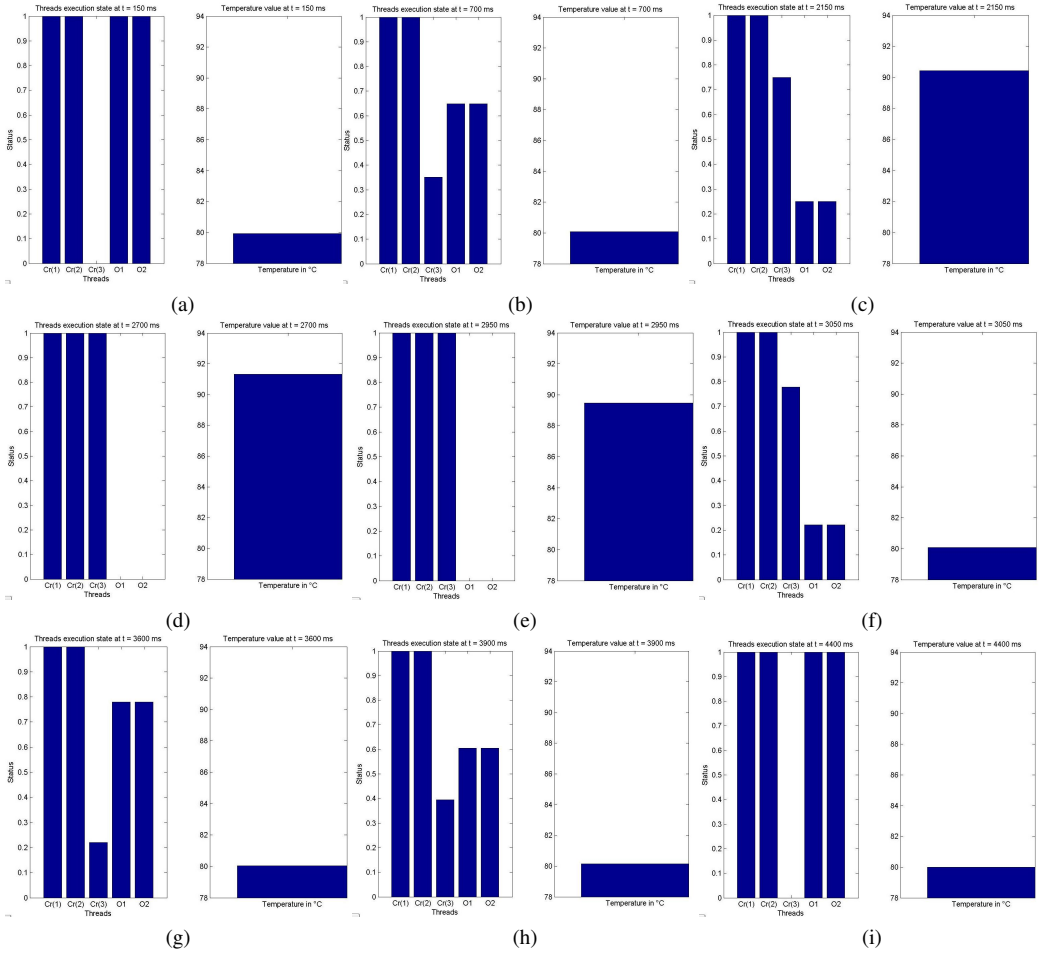


Figure 7. Threads's execution state according to the temperature at specific simulation points.

the proper components' functionalities and the communication among them on the one hand, the implemented accurate and reliable execution chronology of temporal decoupled processes used to encapsulate them, the environment achieves fine timing resolutions and ensures the functionality described in the reference architecture. As the third and final part of this paper, we presented the model was used to build a prototype of a single-node self-aware and self-expressive system presenting a novel concept for fault-tolerance in avionics systems could be built with the model using the test environment. The simulation results have also been presented and discussed.

Ongoing work is devoted to the optimization of the implementation of the presented environment and the development of more novel fault-tolerance concepts and approaches that are better suitable to the next generation of computing systems, systems with self-properties, and can lessen the performance and functionality restraining high redundancy level of safety-critical systems, most particularly of avionics embedded systems.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Seventh Framework Program under grant agreement no 257906.

REFERENCES

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, 2003, pp. 41–50.

[2] epics, "EPiCS Project," Jan 2014. [Online]. Available: http://www.epics-project.eu/

[3] sapere, "SAPERE Project," Mar 2014. [Online]. Available: http://www.sapere-project.eu/

[4] recognition, "Recognition Project," Mar 2014. [Online]. Available: http://www.recognition-project.eu/

[5] S. Parsons, R. Bahsoon, P. R. Lewis, and X. Yao, "Towards a better understanding of self-awareness and self-expression within software systems," University of Birmingham, School of Computer Science, UK, Tech. Rep. CSR-11-03, Apr 2011.

[6] P. R. Lewis et al., "A survey of self-awareness and its application in computing systems," in Proc. Int. Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW). IEEE Computer Society, 2011, pp. 102–107.

[7] T. Becker et al., "EPiCS: Engineering proprioception in computing systems," in Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on, 2012, pp. 353–360.

[8] I. C. Society, IEEE Standard for Standard SystemC Language Reference Manual - IEEE Std 1666^{TM}-2011, 2012.

[9] F. Kesel, Modeling of digital Systems with SystemC: From the RTL- to the Transaction-Level-Modeling. Oldenbourg Wissenschaftsverlag, 2012.

[10] R. Orsagh, D. Brown, P. Kalgren, A. Byington, C.S. ; Hess, and T. Dabney, "Prognostic health management for avionic systems," in Aerospace Conference, IEEE , 2006, pp. 1213–1219.

[11] M. Pignol, "COTS-based applications in space avionics," in DATE 2010, 2010, pp. 1213–1219.

 ISBN: 978-1-61208-341-4

Automated Fault Analysis and Filter Generation for Adaptive Cybersecurity

David J. Musliner, Scott E. Friedman, Jeffrey M. Rye
Smart Information Flow Technologies (SIFT)
Minneapolis, USA
email: {dmusliner,sfriedman,jrye}@sift.net

***Abstract*—We are developing the FUZZBUSTER system to automatically identify software vulnerabilities and create adaptations that shield or repair those vulnerabilities before attackers can exploit them. Adaptive cybersecurity involves efficiently improving software security to minimize the window of attack, and also preserving software functionality as much as possible. This paper presents new tools that have been integrated into FUZZBUSTER adaptive cybersecurity. These tools produce more general, accurate adaptations, increase the efficiency of FUZZBUSTER's diagnoses and adaptation operations, and preserve the software's functionality. We report the results of FUZZBUSTER's analysis of 16 fault-injected command-line binaries and six previously known bugs in the Apache web server. We compare results over different configurations of FUZZBUSTER to characterize the benefits of the new fuzz-testing tools.**

Keywords-cyber defense; automatic filter generation.

I. INTRODUCTION

Cyber-attackers constantly threaten today's computer systems, increasing the number of intrusions every year [1], [2]. Firewalls, anti-virus systems, and patch distribution systems react too slowly to newfound "zero-day" vulnerabilities, allowing intruders to wreak havoc. We are investigating ways to solve this problem by allowing computer systems to automatically identify their own vulnerabilities and adapt their software to shield or repair those vulnerabilities, before attackers can exploit them. Such adaptations must balance the safety of the system against its functionality: the safest behavior might be to simply turn the power off or entirely disable vulnerable applications, but that would make the systems useless. To make a finer-grained balance between security and functionality, adaptations must be:

- General enough to shield the entire vulnerability (i.e., not just blocking an overspecific set of faulting inputs).
- Specific enough to minimize the negative impact on program functionality (e.g., by causing incorrect results on valid inputs).
- Efficiently-generated, to minimize the window of exposure to vulnerability over time.

These considerations for adaptive cybersecurity pose several challenges, including: how faults are discovered and diagnosed, with and without direct access to source code or binaries; how adaptations are generated from the diagnoses; how the many possible adaptations are assessed and chosen; and how all of these operations are orchestrated for efficiency.

This paper describes strategies for automatically discovering vulnerabilities, diagnosing them, and adapting programs to defend against them. We have implemented these strategies within the FUZZBUSTER integrated system for active cybersecurity [3], which includes metrics [4], and metacontrol [5] for self-adaptative software immunity. FUZZBUSTER uses a diverse set of custom-built and off-the-shelf fuzz-testing tools and code analysis tools to develop protective self-adaptations. Fuzz-testing tools find software vulnerabilities by exploring millions of semi-random inputs to a program. FUZZBUSTER also uses fuzz-testing tools to refine its models of known vulnerabilities, clarifying which types of inputs can trigger a vulnerability. FUZZBUSTER's behavior falls into two general classes, as illustrated in Figure 1:

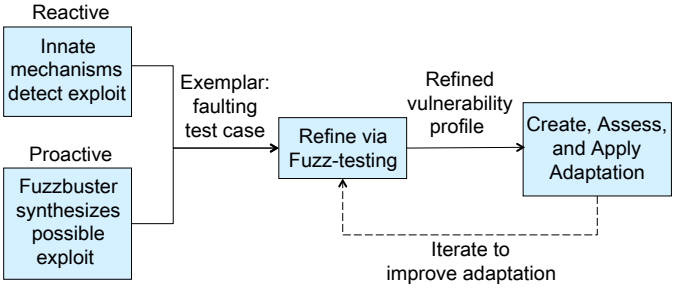


Fig. 1. FUZZBUSTER automatically finds vulnerabilities, refines its understanding of their extent, and creates adaptations to shield or repair them.

1) *Proactive*: FUZZBUSTER discovers novel vulnerabilities in applications using fuzz-testing tools. FUZZBUSTER refines its models of the vulnerabilities and then repairs them or shields them before attackers find and exploit them.
2) *Reactive*: FUZZBUSTER is notified of a fault in an application (potentially triggered by an adversary). FUZZBUSTER subsequently tries to refine the vulnerability and repair or shield it against attackers. Reactive vulnerabilities pose a greater threat to the host, since these may indicate an imminent exploit by an attacker.

FUZZBUSTER's primary objective is to protect its host by adapting its applications, but this may come at some cost. For example, applying an input filter or a binary patch may create a new vulnerability, re-enable a previously-addressed vulnerability, or otherwise negatively impact an application's usability by changing its expected behavior. This illustrates a tradeoff between functionality and security, and measuring both of these factors is important for making decisions about adaptive cybersecurity.

We begin by outlining FUZZBUSTER's process of discovering, refining, and repairing vulnerabilities in Section II,

 ISBN: 978-1-61208-341-4

which motivates our research on adaptation metrics. We then describe FUZZBUSTER's novel diagnosis tools for adaptive cybersecurity in Section III, and we summarize the results of several experiments in Section IV.

II. BACKGROUND: FUZZBUSTER ACTIVE CYBERSECURITY

FUZZBUSTER tests and adapts multiple applications on a host machine. When FUZZBUSTER discovers a fault in one of these applications— or when it is notified of a *reactive* fault triggered by some other input source— it represents the fault as an *exemplar* that contains information about the system's state when it faulted, as shown in Figure 1. Note that FUZZBUSTER is not responsible for fault detection; we assume that other security and correctness mechanisms detect the fault and notify FUZZBUSTER.

An exemplar includes information for replicating the fault, such as environment variables and data passed as input to the faulting application (e.g., via sockets or `stdin`). Some of this data may be unrelated to the underlying vulnerability. For instance, when FUZZBUSTER encounters a fault in the Apache web server in Section IV, it captures all environment variables (all of which are unnecessary to replicate the fault), and the entire string of network input that was sent to the application (most of which is unnecessary to replicate the fault). FUZZBUSTER uses fuzz-testing tools to incrementally refine the exemplar, trying to characterize the minimal inputs needed to trigger the fault. Since time and processing power is limited, FUZZBUSTER uses a greedy meta-control strategy [5] to orchestrate these tools.

Refinement is an iterative process, where each task improves the *vulnerability profile* that FUZZBUSTER uses to characterize the vulnerability. The refinement process turns the initial (often over-specific) vulnerability profile into a more accurate and general profile. While refining the Apache web server vulnerabilities, FUZZBUSTER uses an environment variable fuzzer to test and remove unnecessary environment variables for replicating the fault, uses input fuzzers to delimit, test, and remove/replace unnecessary network input, and thereby develops a more accurate vulnerability profile.

FUZZBUSTER has several general adaptation capabilities, including input filters, environment variable filters, and source-code repair and recompilation. These protect against entire classes of exploits that may be encountered in the future. FUZZBUSTER uses each of these by (1) constructing the adaptation, (2) assessing the adaptation by temporarily applying it for test runs, and (3) applying the adaptation to the production application if it is deemed beneficial. FUZZBUSTER may apply multiple adaptations to an application to repair a single underlying vulnerability. In the case of adapting the Apache web server in Section IV, FUZZBUSTER creates input filters based on its vulnerability profiles: it extracts regular expressions that characterize the pattern of faulting inputs, including necessary character sequences (e.g., "Cookie:"), length-dependent wildcards (e.g., ".{256,}?"), and more. FUZZBUSTER then uses these input filters to identify potentially-faulting inputs and then discard them or rectify them, based on the application under test.

A. Assessing Adaptations

FUZZBUSTER cannot blindly apply adaptations, since they might have a negative impact on functionality or, even worse, they could create new faults altogether. Thus, FUZZBUSTER uses concrete metrics to assess the impact of candidate adaptations on security and functionality.

FUZZBUSTER's adaptation metrics are based on *test cases*: mappings from application inputs (e.g., sockets, `stdin`, command-line arguments, and environment variables) to application outputs (e.g., `stdout` and return code). A *faulting test case* terminates with an error code or its execution time exceeds a set timeout parameter, while a *non-faulting test case* terminates gracefully. FUZZBUSTER stores three sets of test cases for each application under its control:

1) *Non-faulting (reference) test cases* are test cases that were supplied with an application for regression testing. FUZZBUSTER tracks which of these have correct behavior (i.e., output and return code), and which have different/incorrect behavior, given some adaptations.
2) *Faulting test cases* include exemplars that caused faults on their first encounter, and other faulting test cases encountered while refining the exemplar. FUZZBUSTER tracks which of these have been fixed by the adaptations created so far, and which are still faulting. There are two specific types of faulting test cases:
   a) *Reactive faulting test cases*: encountered by host notification and subsequent refinement (see Figure 1). These pose more of a threat, since the underlying vulnerability may have been caused by an adversary.
   b) *Proactive faulting test cases*: encountered by discovery and refinement (see Figure 1). These pose less threat, since they were discovered internally and FUZZBUSTER has no evidence that an adversary is aware of them.

We can calculate two important metrics from these sets of test cases over time:

1) *Exposure* is computed as the number of unfixed faulting test cases over time. This represents an estimated window of exploitability.
2) *Functionality loss* is computed as the number of incorrect non-faulting (reference) test cases over time. This represents the usability that FUZZBUSTER has sacrificed for the sake of security.

Before FUZZBUSTER has discovered faults or been notified of faults, there are no faulting test cases for any application. As FUZZBUSTER encounters proactive and reactive faults and refines those faults (e.g., by experimenting with different inputs), it will accrue faulting test cases. FUZZBUSTER then applies and removes adaptations to fix these faulting test cases. These adaptations ultimately protect the host against adversaries.

 ISBN: 978-1-61208-341-4

FUZZBUSTER's assessment policy allows it to sacrifice functionality to fix faulting test cases. The exact balance can be tuned for different applications, but FUZZBUSTER's default priorities are:

1) Fixing reactive faulting test cases.
2) Fixing proactive faulting test cases.
3) Maintaining the behavior of non-faulting test cases.

This means that FUZZBUSTER will tolerate functionality loss (i.e., by changing the behavior of non-faulting test cases) in order to decrease exposure.

B. Pre-existing Tools for Discovery & Refinement

Since this paper presents new tools for discovery and refinement (Section III), for the sake of comparison we first review the set of fuzz tools we used in previous work [5], [3], [4]. Those tools included a random string generator for discovering faults (called `Fuzz-2001`) and various minimization (i.e., unnecessary character removal) tools for refining faults.

`Fuzz-2001` quickly constructs a sequence of printable and non-printable characters and feeds it as input to the program under test. This is effective for discovering some buffer overflows, problems with escape characters, and other such problems.

The minimization tools FUZZBUSTER uses to refine vulnerabilities include:

- *smallify*: semi-randomly removes single characters from the input string.
- *line-relev*: semi-randomly removes entire lines from the input string.
- *divide-and-conquer*: Use a binary search to attempt to remove entire portions of the input string.

Each of these tools is designed take a faulting test case as input, and produce smaller faulting test case(s).

Minimization tools can operate in a black-box fashion, where FUZZBUSTER does not have the source code or even access to the binary. All they require is an output signal to determine whether the program faulted.

III. NEW DISCOVERY & REFINEMENT TOOLS

We now discuss several new tools that we have incorporated into FUZZBUSTER for discovering and refining faults. We then present empirical results comparing the new and existing tools to characterize the effects on the host's exposure to vulnerabilities.

Both of these tools work with *input filter adapations*; that is, program adaptations that remove content from input data before passing the data to the corresponding program.

A. Retrospective Fault Analysis

We implemented and tested *Retrospective Fault Analysis (RFA)*, a new tool for vulnerability discovery. RFA works by finding the most recent faulting test case such that:

- The test case's input is filtered by the most recent adaptation applied, so some input data has been removed.
- The test case still faults, despite its input being filtered.

RFA then uses the test case— with filtered input— as an exemplar. This effectively allows FUZZBUSTER to fix test cases that still fault, despite incremental adaptations.

To illustrate why this is important, consider the following simplified example, where a program faults if it receives either `CRASH` or `fault` in an incoming message. Some messages may have more than one fault within them, e.g.:

- `Cookie: foo=...CRASH...fault...`
- `Cookie: foo=...faCRASHult...`

This means that FUZZBUSTER can automatically build a filter adaptation to address `CRASH`, but in both of the above cases, there will still be a `fault`. Using RFA, FUZZBUSTER will follow its `CRASH` adaptation with a retrospective investigation of the remaining `fault` test case(s). This produces a more complete analysis of problematic inputs, and it improves the host's exposure to vulnerabilities, as we demonstrate in our experiments.

B. Input Generalization Tools

As described in Section II-B, minimization tools remove unnecessary characters for a fault. Unfortunately, refining vulnerabilities based on removal alone will tend to produce overspecific adaptations.

Consider the example of IP addresses within a packet header: minimization tools might trim 192.168.0.1 to 2.8.0.1, which might still produce the fault; however, an adaptation based on this model will only be effective when 2, 8, 0, and 1 are all present in the address.

FUZZBUSTER's new generalization tools go the extra step of replacing characters and inserting characters to generalize FUZZBUSTER's regular expression model of the faulting input pattern. This means that FUZZBUSTER will be able to substitute the IP address' digits with other digits to develop a more general, accurate adaptation.

We have implemented the following generalization tools:

- `replace-all-chars`: replaces all characters with different characters, reruns the test case, and then generalizes. This determines whether the test case is an instance of a buffer overflow. For example:

  `ABCDEFGH ==> .{8,}`
- `replace-delimited-chars`: splits the input into chunks, using common delimiters, removes and replaces delimited chunks, and then generalizes. For example:

  `host: 1.1.1.1\nCookie ==> .{0,}?Cookie`
- `replace-individual-chars`: removes and replaces individual characters, sensitive to character classes (e.g., letters, digits, whitespace, etc.), and generalizes. For example:

  `GCOJR34A59S94H ==> .*C.*R.*A.*S.*H`
- `insert-chars`: inserts characters in-between consecutive concrete characters, to relax adjacency constraints. For example:

  `CRASH ==> .*C.*R.*A.*S.*H`

 ISBN: 978-1-61208-341-4

- `shorten-regex`: removes characters within wildcard blocks to provide more accurate buffer overflow thresholds. For example:

  `host: .{951,} ==> host: .{256,}`

We conducted experiments on multiple programs to characterize the effect of generalization tools and RFA. We discuss these experiments and results next.

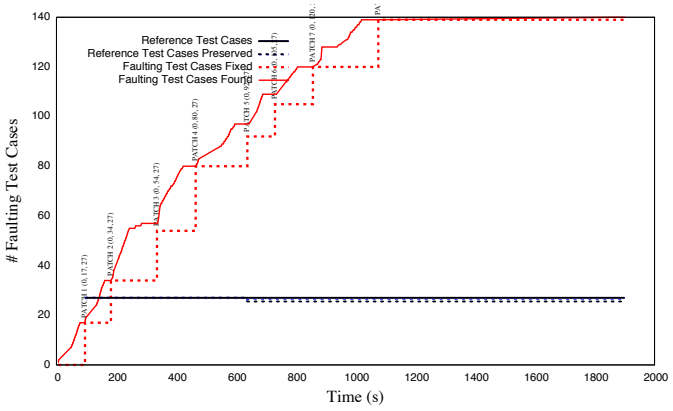


Fig. 2. Results using RFA, minimization, and generalization.

IV. Experiments

We conducted an empirical evaluation on different programs to measure the effect of RFA and the new generalization fuzztools. We divide this into four discussions: (1) a comparative analysis of minimization, generalization, and RFA on a single program; (2) an example of FUZZBUSTER sacrificing functionality in order to increase security; (3) a quantitative comparison of minimization and generalization using FUZZBUSTER to shield a web server against known vulnerabilities; and (4) adaptation statistics across multiple programs using FUZZBUSTER with generalization and RFA.

A. Comparative Analysis: Generalization, Minimization, RFA

For this experiment, we used a fault-injected version of `dc`, a unix-based, `stdin`-based desktop calculator program. The fault in `dc` was injected within the internal modulo (i.e., remainder) operation. This operation is reached by invoking the `%` command with at least two numbers on the stack, printing with a non-decimal output radix, changing the input radix, or invoking base conversion.

We ran FUZZBUSTER in five settings: with RFA using *both* minimization and generalization tools (Figure 2); and then with and without RFA, under either minimization or generalization tools (Figure 3).

Each of these plots display the following important data for adaptive cybersecurity:

- The number of faulting test cases FUZZBUSTER has identified through discovery and refinement (solid light red line).
- The number of those faulting test cases that FUZZBUSTER has fixed (dashed light red line).
- Exposure to vulnerabilities (area between light red lines).
- The number of reference (non-faulting) test cases FUZZBUSTER has for the application (solid dark line).
- The number of those non-faulting test cases whose return code and output behavior is preserved in the patched version (dashed dark line).
- Loss of functionality (area between dark lines).
- The patches that have been applied.

The comparison plots in Figure 3 illustrate the tradeoffs of generalization and RFA. Minimization tools (Figure 3, left) produce quick, overspecific patches. For instance, **PATCH 16** in the Figure 3 upper-left plot filters the pattern `.*9.*5.*%.*`. While this is a legitimate example of the fault, it does not characterize the fault in its entirety. By comparison, the generalization patches are slightly more general.

Figure 3 also illustrates the effect of retrospective fault analysis. In the RFA trials, the exposure (distance between the light red lines) is significantly reduced. This is because FUZZBUSTER often deploys a filter that addresses some – but not all – problems in a faulting input, and then RFA allows FUZZBUSTER to focus on the remainder of the problematic input. For instance, if a single test case has both a modulo operation and a base conversion, filtering out only one of these operations will not repair the test case.

In the setting with both generalization and RFA, FUZZBUSTER filters against the entire vulnerability within 15 minutes; in the other cases, FUZZBUSTER does not level off for over three hours.

Note that in all settings in Figure 3, FUZZBUSTER did not lose functionality of the underlying application, as measured by the correctness of the reference test cases.

Figure 2 shows the results of FUZZBUSTER with both minimization and genralization enabled. It fixes the entire vulnerability and levels off in 18 minutes, but it also destroys the functionality of one of the reference test cases, since its **PATCH 5** was overgeneral.

B. Sacrificing Functionality to Increase Security

We ran another FUZZBUSTER trial on a different fault-injected version of the `dc` binary. This version faulted whenever an arithmetic operation is invoked on an empty stack, so for instance, the sequence ``9 5 +'' would not fault, but the inputs ``+'' or ``4 n +'' would fault due to an empty stack (and ``n'' pops the stack).

The results are shown in Figure 4. Using generalization tools and RFA, FUZZBUSTER isolates individual arithmetic operations and generates filters for each, ultimately disabling its arithmetic operations to prevent any faults. Note that almost every adaptation has an adverse impact on program functionality, but by design, these are acceptable losses to increase safety of the host.

C. Adapting a Web Server

We conducted FUZZBUSTER experiments on known Common Vulnerabilities and Exposures (CVEs) on the Apache web server. This demonstrates FUZZBUSTER working on larger

 ISBN: 978-1-61208-341-4

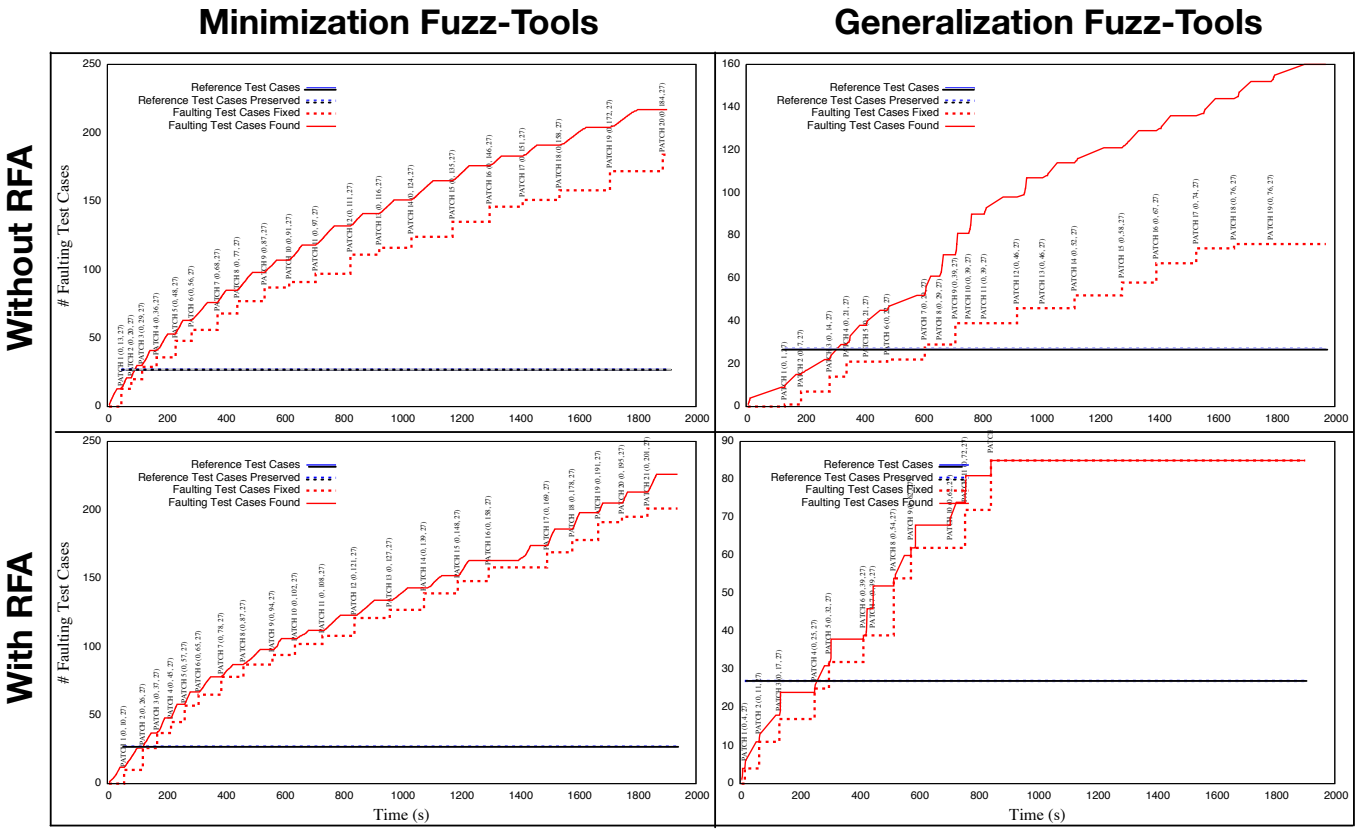


Fig. 3. Results comparing the exposure window of Retrospective Fault Analysis and minimization vs. generalization tools.

production-quality applications with real vulnerabilities, and it shows the generality of FUZZBUSTER and its fuzz-tools.

For each trial, we initialized FUZZBUSTER with the Apache web server as the only application under test. We then sent a faulting message to the server— as dictated by the corresponding CVE— and FUZZBUSTER detected the reactive fault and began its fuzzing. Table I reports how many minutes FUZZBUSTER took to produce an input filter adaptation (from simulation start to patch time) for the corresponding CVE using only minimization tools (i.e., "Min."), only generalization tools (i.e., "Gen."), and the speedup provided by generalization tools.

TABLE I
FUZZBUSTER'S REACTION TIME ON CVES OF THE APACHE WEB SERVER.

CVE	RT (Min.)	RT (Gen.)	Speedup
2011-3192	96	4	24x
2011-3368-1	53	10	5x
2011-3368-2	32	10	3x
2011-3368-3	77	11	7x
2012-0021	36	3	12x
2012-0053	30	7	4x

Reaction times reported in minutes; speedup reported as quotient.

In addition to producing more general patches, the generalization tools also yield a significant speedup factor between 3x and 24x, and on average, produce useful adaptations in an order of magnitude less time.

For these CVE trials, RFA was not necessary since FUZZBUSTER fixes all faulting test cases with the first patch it produces.

D. Statistics Across Programs

We now present additional results from using FUZZBUSTER with the generalization tools and retrospective fault analysis on 16 fault-injected binaries.

We used GenProg [6], an evolutionary program repair tool, to create faulty binaries from the source code of unix command-line applications including `dc`, `fold`, `uniq`, and `wc`. We achieved this by modifying the GenProg test cases—which GenProg uses as a fitness function— to expect a fault on certain inputs. This way, GenProg would generate selectively-faulting binaries based on our specifications.

FUZZBUSTER automatically analyzed each faulty binary for two hours, using a mix of proactive fuzz-tools (e.g., `Fuzz-2001` and `RFA`), refinement fuzz tools (e.g., the generalization fuzz-tools), and adaptation strategies (e.g., input filters).

Fuzzing *leveled off* (i.e., FUZZBUSTER patched the entire injected fault, based on our manual analysis of patches) on 10/16 binaries. Of these leveled-off binaries, FUZZBUSTER took an average of 5.87 minutes to level off, and it sacrificed an average of 6% functionality (i.e., by changing the output of

 ISBN: 978-1-61208-341-4

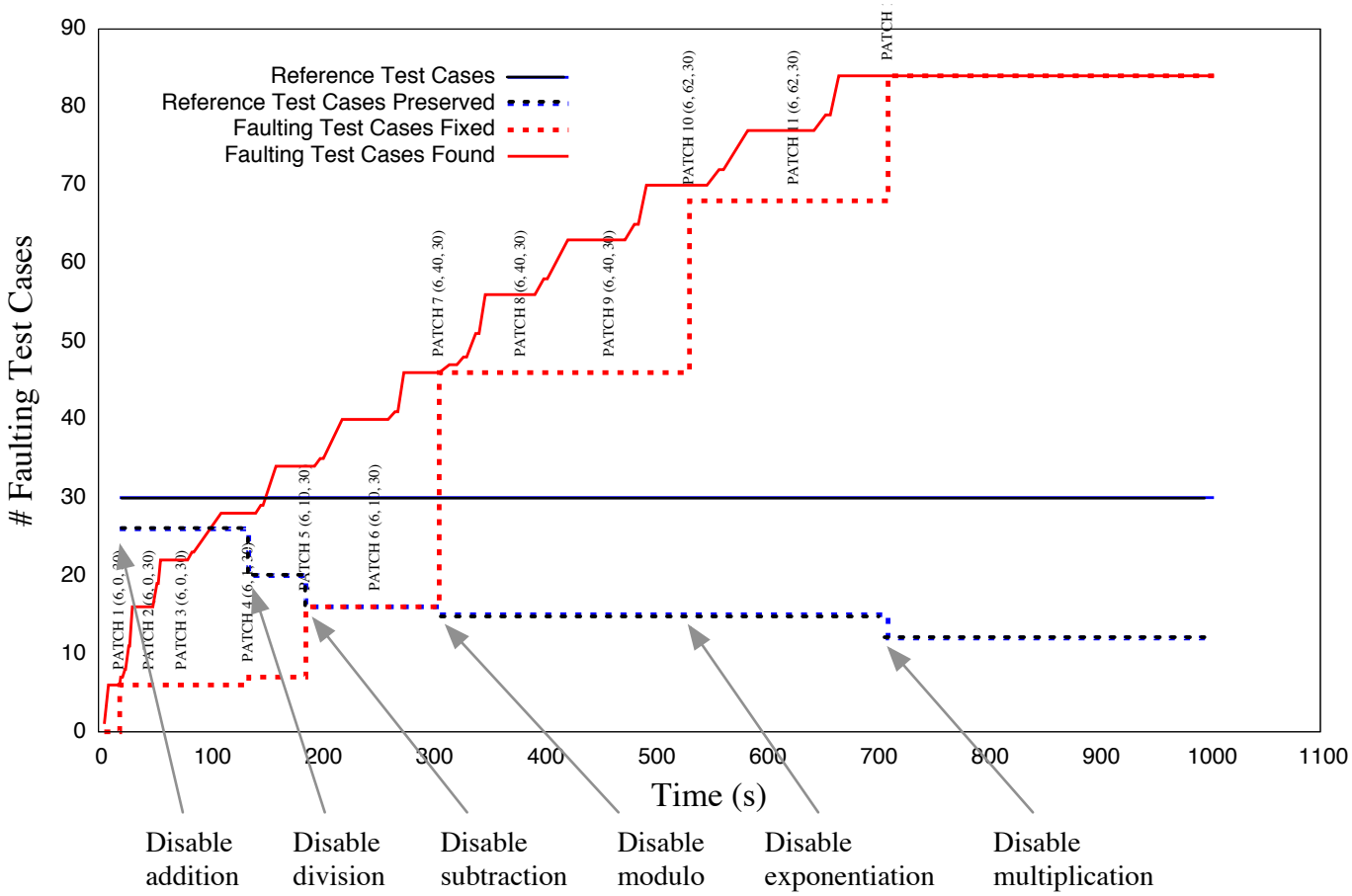


Fig. 4. FUZZBUSTER sacrifices functionality to protect the program against vulnerabilities.

non-faulting reference test cases). FUZZBUSTER retained full functionality on 7 of the 10 leveled-off binaries.

Over all 16 fault-injected binaries, FUZZBUSTER created an average of 8.2 adaptations and applied an average of 7.8, which amounts to a 95% usage of the adaptations it created. Over all binaries, FUZZBUSTER fixed an average of 82% of the faulting test cases and sacrificed an average of 10% functionality during each 2-hour trial. This suggests that when FUZZBUSTER cannot generate a perfect adaptation, it still manages to close the exposure window over time.

V. RELATED WORK

As previously noted, the FUZZBUSTER approach has roots in fuzz-testing, a term first coined in 1988 applied to software security analysis [7]. It refers to invalid, random or unexpected data that is deliberately provided as program input in order to identify defects. Fuzz-testers— and the closely related "fault injectors"— are good at finding buffer overflow, cross-site scripting, denial of service (DoS), SQL injection, and format string bugs. They are generally not highly effective in finding vulnerabilities that do not cause program crashes, e.g., encryption flaws and information disclosure vulnerabilities [8]. Moreover, existing fuzz-testing tools tend to rely significantly on expert user oversight, testing refinement and decision-making in responding to identified vulnerabilities.

FUZZBUSTER is designed both to augment the power of fuzz-testing and to address some of its key limitations. FUZZBUSTER fully automates the process of identifying seeds for fuzz-testing, guides the use of fuzz-testing to develop general vulnerability profiles, and automates the synthesis of defenses for identified vulnerabilities.

To date, several research groups have created specialized self-adaptive systems for protecting software applications. For example, both AWDRAT [9] and PMOP [10] used dynamically-programmed wrappers to compare program activities against hand-generated models, detecting attacks and blocking them or adaptively selecting application methods to avoid damage or compromises.

The CORTEX system [11] used a different approach, placing a dynamically-programmed proxy in front of a replicated database server and using active experimentation based on learned (not hand-coded) models to diagnose new system vulnerabilities and protect against novel attacks.

While these systems demonstrated the feasibility of the self-adaptive, self-regenerative software concept, they are closely tailored to specific applications and specific representations of program behavior. FUZZBUSTER provides a general approach

 ISBN: 978-1-61208-341-4

to adaptive immunity that is not limited to a single class of application. FUZZBUSTER does not require detailed system models, but will work from high-level descriptions of component interactions such as APIs or contracts. Furthermore, FUZZBUSTER's proactive use of intelligent, automatic fuzz-testing identifies possible vulnerabilities before they can be exploited.

VI. CONCLUSION AND FUTURE WORK

FUZZBUSTER is designed to discover vulnerabilities and then quickly refine and adapt its applications to prevent them from being exploited by attackers. This paper presented two advances in FUZZBUSTER's tools — retrospective fault analysis and generalization fuzz-tools — aimed at improving the quality and efficiency of FUZZBUSTER's adaptations. We presented empirical results of FUZZBUSTER's automated analysis of fault-injected programs and real CVEs, using objective metrics for adaptive cybersecurity such as vulnerability exposure, functional loss, and reaction time. When analyzing fault-injected programs, the generalization fuzz-tools and RFA reduced vulnerability exposure by a factor of five on fault injected programs, and allowed FUZZBUSTER to filter out more of the vulnerability in less time. When analyzing the Apache HTTP server, the fault generalization tools yielded an order of magnitude speedup in reaction time over the existing fault minimization tools.

At present, FUZZBUSTER uses a wrapper around the programs it controls, and its wrapper filters all incoming data according to the current adaptations (e.g., input filters) before sending the data to the binary. One next step is to revise the program's binary directly, and embed the input filters as preprocessors.

The generalization fuzz-tools and RFA are all domain-general strategies, and we demonstrated this by using them to improve program analysis on command-line filter programs (e.g., `wc`), state-dependent standard input programs (e.g., `dc`), and grammar-specific web programs (e.g., Apache HTTP server). The most domain-specific enhancement is the `replace-delimited-chars` tool that uses common delimiters to analyze portions of data. This tool contributed significantly to the speedup of FUZZBUSTER's analysis of HTTP headers in the Apache HTTP server experiment. We believe that we will see additional performance benefits by adding more domain-specific structures to FUZZBUSTER, including input grammars (e.g., packet header structure) and deeper application models (e.g., recording application command-line options and values).

We anticipate using the adaptive cybersecurity metrics from this paper (see also [5], [4]) to evaluate future design decisions for FUZZBUSTER and other active cybersecurity projects.

ACKNOWLEDGMENTS

This work was supported by DARPA and Air Force Research Laboratory under contract FA8650-10-C-7087. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for public release, distribution unlimited.

REFERENCES

[1] T. Kellerman, "Cyber-threat proliferation: Today's truly pervasive global epidemic," *Security Privacy, IEEE*, vol. 8, no. 3, pp. 70 –73, May-June 2010.

[2] G. C. Wilshusen, "Cyber threats and vulnerabilities place federal systems at risk: Testimony before the subcommittee on government management, organization and procurement," United States Government Accountability Office, Tech. Rep., May 2009.

[3] D. J. Musliner, J. M. Rye, D. Thomsen, D. D. McDonald, and M. H. Burstein, "FUZZBUSTER: A system for self-adaptive immunity from cyber threats," in *Eighth International Conference on Autonomic and Autonomous Systems (ICAS-12)*, March 2012.

[4] D. J. Musliner, S. E. Friedman, T. Marble, J. M. Rye, M. W. Boldt, and M. Pelican, "Self-adaptation metrics for active cybersecurity," in *SASO-13: Adaptive Host and Network Security Workshop at the Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, September 2013.

[5] D. J. Musliner, S. E. Friedman, J. M. Rye, and T. Marble, "Meta-control for adaptive cybersecurity in FUZZBUSTER," in *Proc. IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems*, sep 2013.

[6] W. Weimer, T. Nguyen, C. L. Goues, and S. Forrest, "Automatically finding patches using genetic programming," *Software Engineering, International Conference on*, vol. 0, pp. 364–374, 2009.

[7] B. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, December 1990.

[8] C. Anley, J. Heasman, F. Linder, and G. Richarte, *The Shellcoder's Handbook: Discovering and Exploiting Security Holes, 2nd Ed.* John Wiley & Sons, 2007, ch. The art of fuzzing.

[9] H. Shrobe, R. Laddaga, B. Balzer, N. Goldman, D. Wile, M. Tallis, T. Hollebeek, and A. Egyed, "AWDRAT: a cognitive middleware system for information survivability," *AI Magazine*, vol. 28, no. 3, p. 73, 2007.

[10] H. Shrobe, R. Laddaga, B. Balzer *et al.*, "Self-Adaptive systems for information survivability: PMOP and AWDRAT," in *Proc. First Int'l Conf. on Self-Adaptive and Self-Organizing Systems*, 2007, pp. 332–335.

[11] "Cortex: Mission-aware cognitive self-regeneration technology," Final Report, US Air Force Research Laboratories Contract Number FA8750-04-C-0253, March 2006.

 ISBN: 978-1-61208-341-4

Sensor-Hub: A Real-Time Data Integration and Processing Nexus for Adaptive C2 Systems

Jean-François Gagnon, Sébastien Tremblay
School of Psychology
Université Laval
Québec, Canada
e-mail: jean-francois.gagnon.22@ulaval.ca
sebastien.tremblay@psy.ulaval.ca

Daniel Lafond, Martin Rivest, François Couderc
Thales Research and Technology Canada
Thales Canada
Québec, Canada
e-mail: daniel.lafond@ca.thalesgroup.com
martin.rivest@ca.thalesgroup.com
francois.couderc@ca.thalesgroup.com

***Abstract*—The present paper introduces the Sensor-Hub, a prototype tool for augmenting the common operating picture and adaptability of distributed teams in safety-critical environments. The Sensor-Hub aims to facilitate the integration and interpretation of data collected directly from humans augmented with sensing capability involved in the situation to produce timely and relevant information on the current functional state of operators, the situation and their environment. Herein, we elaborate on the development and validation of the sensing and interpretation framework, emphasising the key adaptation capabilities that it seeks to enable. Lastly, this paper illustrates three sectors of application of the Sensor-Hub: training of safety-critical team operations, real-time error-prevention and adaptation during operations, and assessment of inter-agent and human-technology interactions.**

Keywords-Augmenting humans with technology; sensing; modeling; situation awareness; network centric operations.

I. INTRODUCTION

Sensing technologies have evolved to the point that valuable information on an evolving operation can not only be acquired with regards to observable characteristics of the environment, but also about the functional state of the team during the accomplishment of its mission [1][2][3]. Advanced human sensing tools provide the opportunity to increase decision making and situational awareness of personnel actively engaged in a task and their immediate environment. Applications exploiting such data have the potential to significantly improve individual and team situational awareness, safety, adaptability and performance – provided that the data is processed by valid and reliable assessment models.

Situational awareness is defined as the perception of information pertaining to a situation, the comprehension of its meaning, and the projection of the situation into a near-future [4][5]. It is widely posited that situational awareness is associated with operational success, as it is the basis for a Common Operating Picture (COP) within the response team. Unfortunately, the characteristics of many situations severely hinder the perception of data, its comprehension and consequently its projection into the future. This is the case, for instance, during emergency response. Emergency response is the active phase of emergency management at the occurrence of an incident. Team members involved in emergency response must coordinate their effort in order to perform effectively often in very stressful, life-threatening environments, despite the challenges of the geographically distributed nature of their work (e.g., the command center is frequently delocalized from the incident and first responders are often dispersed across the area of operations). The uncertainty and time pressure that characterise emergency response situations often severely constrains the quality of the COP, both at the tactical and operational levels. Moreover, the amount of information to consider is often considerable, pushing individuals' cognitive capacities to their limit. The state of readiness of different team members can be particularly difficult to assess in such contexts.

Despite these constraints, there is a growing effort to augment accessibility and reliability of information in these (or similar) environments [6]. The deployment of multiple sensors in these environments enables the application of a network-centric approach to operations. The concept of Network-Centric Operations (NCO) originates from the military domain and refers to linking networks of sensors, decision makers, and individual agents [7][8] to achieve information superiority. This approach strives to increase shared awareness, self-synchronization, and performance of the network as a whole [9]. Concepts pertaining to network-centric operations can also be applied to emergency response, as many similarities exist between the two domains [10]. The objective of this paper is to discuss on how the developments in human sensing can take the NCO approach to a new level by enabling adaptive solutions to Command and Control (C2) in complex, distributed environments. To serve this purpose, we use the Sensor-Hub concept as a demonstrator of potential increased capabilities in the context of C2 and emergency response.

 ISBN: 978-1-61208-341-4

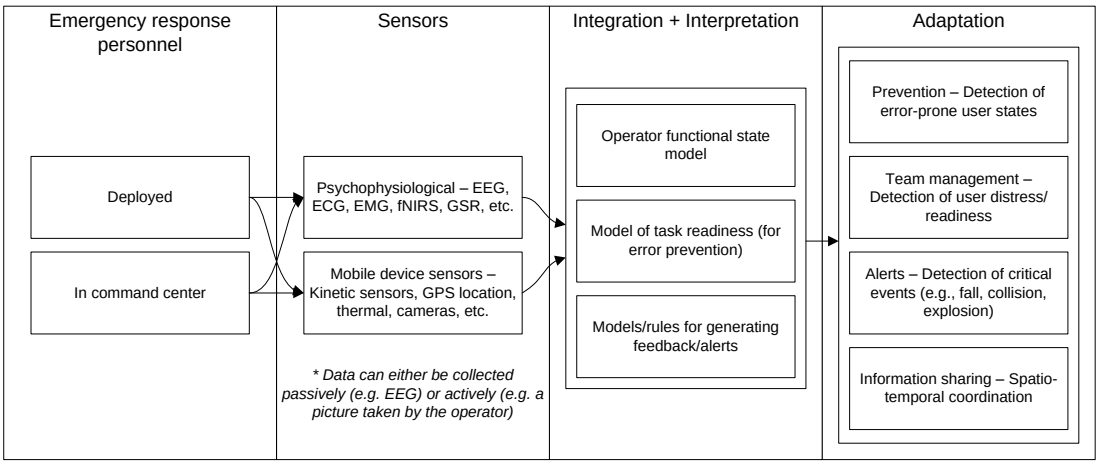


Figure 1. Sensor-Hub Framework.

This paper is divided into six sections. Section I sets the general context in which the Sensor-Hub could be deployed and implemented. Section II describes the potential sensing capabilities of the Sensor-Hub. Section III is concerned with the validation and the calibration of the higher-level models that will interpret the signals from the sensors. Section IV highlights the key features of the Sensor-Hub and discusses how these features could facilitate its implementation into different security and military contexts. Section V describes how the interpretation of sensors in the context of C2 and emergency response can trigger adaptive automation such as cognitive counter-measures. Finally, Section VI summarizes the critical components of the Sensor-Hub with regards to the concept of NCO.

II. HUMAN SENSING AND SITUATION MODELING

The Sensor-Hub aims to integrate and interpret data from multiple sensors mounted on emergency response personnel, either stationed in command centers or deployed in the field. Specifically, it integrates data from multiple sensors to derive metrics pertaining to Operators' Functional State (OFS), and passively/actively captures data about the environment. The main innovative component of the Sensor-Hub is its ability to model a series of data inputs (e.g., heart rate variability, velocity, blood pressure, positional data, temperature, etc.) and output relevant information about the functional state of the operator or the state of the operator's immediate environment. The Sensor-Hub is designed with built-in models and decision rules (to be calibrated and validated using experimental data), and allows for specifying relationships/rules as necessary to adjust to new contexts, or to different functional state concepts. The Sensor-Hub builds upon existing sensing technologies by integrating their signal and interpreting the data to derive higher level concepts. Below we describe the modeling framework (Fig. 1).

A. Sensing the Environment

Safety-critical environments are rapidly changing, dynamic and complex, which make them hard to predict. Although different situations (e.g., a toxic gas leak and a residential fire) may require different types of information, the individuals in charge of emergency response will often be well situated to provide the required information to higher level decision makers or other responders arriving on site. Providing sensors to these individuals and allowing the information of these sensors to be integrated and distributed across the response team is a key factor to augment Situation Awareness (SA) in this context and to effectively adapt to the situation. The Sensor-Hub seeks to facilitate integration and sharing of environmental information by allowing tactical operators to capture (either passively or actively) geotagged pictures, video, sounds, and measurements such as pressure and temperature. This data can be distributed across team members, both at the tactical level and command levels to support decision making and create a more capable emergency workforce. The capacity to sense the environment will help provide timely and relevant support for coordination and collaboration between tactical responders.

B. Sensing Operator Functional State (OFS)

Considerable amount of effort is put in the development of systems aiming to monitor OFS – a concept that groups together the mediators of human performance in a given context. OFS is a multidimensional concept that represents the current capacity of an individual to carry out his/her task without errors. As stated by Hockey and Robert [11], OFS is intrinsically defined in relation to the task to be carried out and its associated costs in terms of cognitive and physical efforts. For instance, in remotely operated vehicle piloting tasks, OFS involves the capacity of the pilot to re-allocate his attention between tasks, deemed critical for responding appropriately to alarms. In this context, OFS may also involve fatigue and stress as they are all related to piloting performance. Systems aiming to determine OFS are mostly, if not all, based on the assessment of the psycho and neurophysiologic state of the operator and the interpretation of this signal to derive OFS-related concepts such as fatigue, mental workload and stress. The major value in OFS assessment is the ability to anticipate human error (i.e., recognize states with high error probabilities), allowing the user or team to take preventive action. Mobile assessments

 ISBN: 978-1-61208-341-4

of OFS are becoming increasingly feasible using wearable sensors. Furthermore, kinetic data and vital signs of deployed personnel can provide valuable information for increasing collaboration effectiveness and survivability.

C. Sensing the Situation

The OFS cannot be established solely by monitoring the operator, but requires information about the interactivity of the operator and the tasks; and on how efficient and responsive the operator is at performing his/her tasks [11]. Since there is most often a variety of tasks and sub-task to be carried out by operators in safety-critical environments, the accurate assessment of the OFS is directly related to the capacity to identify ongoing tasks and monitor modulations in performance. Determining the ongoing task remains a challenge, however, innovative techniques like dynamic cognitive task analysis [12] could be used to detect a pattern in the series of actions carried out by the operator and consequently infer the actual task being performed. Although this is not currently implemented in Sensor-Hub, having such a model capable of sensing the situation is key to augmenting the COP.

III. EXPERIMENTAL CALIBRATION/VALIDATION

The work in progress for the development of the integration and interpretation model within the Sensor-Hub involves several steps. First, an initial OFS model, based on existing literature, will be developed. This initial model will specify the filters applied to the incoming sensor data, the nature of the relationships between the different type of data generated by the sensors, and the format of the output generated. For the initial demonstration, the number of sensors used is restrained and will include heart rate monitor, global positioning system, and accelerometer. These sensors are chosen because they can provide several different metrics that were proven to be good indicators of operators' state in previous studies [11]. Heart rate variability, breathing rate, speed and acceleration will most likely be integrated within the OFS model. A hybrid architecture for the OFS model is adopted to allow for flexibility, meaning that it can easily be adapted to fit a wide variety of data with different properties. Both logical rules and the general class of regression models are implemented within the Sensor-Hub. Regression-based models have a demonstrated capacity to fit highly non linear data in similar contexts [13], whereas logical rules allow for quick implementation of disjunctive or conjunctive rules, for instance, which may be harder to model with regressions. Disjunctive rules may allow, for instance, one to differentiate between mental and physical workload based on heart rate. Second, the OFS model will be empirically calibrated through the observation of pilot participants completing an emergency response scenario. The appropriate calibration of the relationship between the data collected by the sensors and the various concepts that the Sensor-Hub aims to monitor (e.g., OFS and critical events in the environment) requires empirical testing in order to ensure good sensitivity and specificity. This involves a realistic emergency response scenario putting together multiple sub-tasks designed to be as close as possible to the real operational context. This scenario is divided into multiple phases with varying levels of mental demand, team coordination demands, physical demands, and environmental states. Given the live nature of the scenario (i.e., emphasis on realism rather than experimental control in simplified conditions), a good deal of noise in the measurement of the aforementioned concepts is to be expected. The modeling approach thus focuses on simplicity and robustness at the cost of raw accuracy. The scenarios are implemented into the SYnRGY simulation platform. As illustrated in Fig.2, the Safety-critical scenarios involving multidisciplinary teams of emergency responders are scripted and recorded (left). The responders' interface (in command centers) is composed of a geographic information system and multiple panels used for decision making, communications and displaying mission critical information (right).

Figure 2. SYnRGY Simulation platform.

The platform allows for human in-the-loop simulations of safety-critical situations for both deployed and command center personnel [14]. These simulations allow the collection of data for model calibration in this context. Thirdly, model validity will be tested using different participants on a similar, but different emergency response scenario. The participants will be emergency management experts to improve the validity of the results. The purpose of this validation is to evaluate the predictive power of the model. Additionally, volunteers participating in the validation scenario will be asked (retrospectively) to rate the level of each of the output concepts at different moments in time. These added constraints will allow further tuning of the model. The developed scenario will enable the calibration and validation of the OFS model, which are human factors related issues, and to test for scalability and latency of the system which are technology-related issues.

IV. TOOLSET

A key feature of the Sensor-Hub is its toolset to facilitate (1) the integration and interpretation of new sensors and (2) visualisation of its outputs. Since OFS modeling is in its infancy, the development of each model is tedious and may represent an important obstacle, especially for non-specialists. The Sensor-Hub aims to facilitate the development of the OFS model by providing its users with pre implemented models and tuning tools. For instance, users can add or remove logical conditions to increase fit between inputs (i.e., data from sensor) and outputs (i.e., OFS). Calibration of the model is also facilitated by the availability of validated scenarios implemented within SYnRGY. These scenarios are specifically designed to vary cognitive demand, teamwork, and physical demand and consequently constitute an efficient calibrating environment. Moreover, data collected through the simulation platform constitutes an important referential database for further model development. In addition to modeling tools, the Sensor-Hub provides a component for visualizing the inputs and outputs of the model which may provide additional insight for model calibration or for decision makers in operational contexts.

V. ADAPTABILITY

Within the suggested framework, the Sensor-Hub interprets raw data to create mission-relevant information, which could serve as critical events for adaptive systems. For instance, in a safety-critical task training context, the Sensor-Hub can provide useful feedback to tactical operators during debriefings or even in real-scale exercises. Real-time assessment of cognitive load, for instance, has been shown to be insightful in the improvement of training [16]. From this point of view, what is adaptive in the system is the team of responders per se rather than the Sensor-Hub. The latter is the enabler, and so provides information required for triggering team adaptation. Adaptive systems, in general, would benefit from inputs from an assessment of the situation, OFS and of the environment that the tactical operators are facing [15]. Such adaptive systems can, for instance, offer assistance to the operators involved in safety-critical missions when detecting critical OFS levels that are likely to lead to critical failures or when detecting an environmental threat. Moreover, because environmental information can be assessed by the Sensor-Hub, the adaptive system could provide tactical operators with additional informational inputs (i.e., the information flow is bi-directional). Finally, this human-sensing and analysis capability may also be useful as an assessment tool for comprehensive assessments of human-technology interactions or team interactions in a research and development context. In the context of NCOs, and particularly emergency response, the Sensor-Hub can provide critical information on the OFS of first responders for real-time adaptation. One of our currently envisioned applications would use the high-level assessment of the functional state to suggest task re-allocation to commanders for timely team management [17]. Task re-allocation is deemed critical for highly dynamic C2 situations such as emergency response [11]. The adaptive component of the system, triggered by the OFS, is not fully automated, leaving the final decision and responsibility in the hands of the commanders.

VI. CONCLUSION

The current paper discussed how human sensing can support the NCO approach to C2 and emergency response by enabling adaptive solutions in dealing with the challenges of complex and dynamic distributed environments. We illustrate how the data from physiological sensors can be interpreted into higher-level concepts and trigger adaptive "support" to the commanders. The Sensor-Hub framework aims to provide an advanced human sensor integration and modeling capability to support a new and unparalleled network-centric emergency response capability. The framework is nonetheless generic and widely applicable to other domains. Three potential applications of the Sensor-Hub would provide adaptive capabilities to teams evolving in safety-critical environments: 1) Training of safety-critical team operations, 2) Real-time error-prevention and adaptation during operations, and 3) Assessment of inter-agent and human-technology interactions. Others have also developed sensor integration tools relevant to OFS measurement [1][2][18]. However, the key differentiator in the development of the Sensor-Hub is its focus on facilitating the assessment and decision making process by giving a simple yet flexible toolset for editing and calibrating the interpretation model, based on psychophysiological theory and on empirical evidence. Such an empirical evidence is also greatly facilitated by the scenarios implemented with the SYnRGY simulation platform.

ACKNOWLEDGMENT

This work was funded by Thales Research and Technology Canada and by an internship fund granted to Jean-François Gagnon by the MITACS Accelerate program.

REFERENCES

[1] R. Matthews, N. J. McDonald, P. Hervieux, P. J. Turner, and M. A. Steindorf, "A wearable physiological sensor suite for

unobtrusive monitoring of physiological and cognitive state," 29th Annual International Conference of the IEEE, pp. 5276-5281, August 2007, doi: 10.1109/IEMBS.2007.4353532

[2] P. Alexandros and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," Systems, Man, and Cybernetics, Part C: IEEE Transactions on Applications and Reviews, vol. 40, pp. 1-12, January 2010, doi: 10.1109/TSMCC.2009.2032660.

[3] K. M. Stanney, et al., "Augmented cognition: An overview," Reviews of human factors and ergonomics, vol. 5, pp. 195-224, June 2009, doi: 10.1518/155723409X448062.

[4] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," Human Factors: The Journal of the Human Factors and Ergonomics Society, vol. 37, pp. 32-64, March 1995, doi: 10.1518/001872095779049543.

[5] R. Rousseau, S. Tremblay, and R. Breton, "Defining and Modeling Situation Awareness: A Critical Review," In S. Banbury and S. Tremblay (Eds), A Cognitive Approach to Situation Awareness: Theory, Measures and Application, Aldershot, UK, Ashgate, pp. 3-21, 2004.

[6] S. Vieweg, A. Hughes, K. Starbird, and L. Palen, "Microblogging during two natural hazards events: What twitter may contribute to situational awareness," Proceedings of ACM Conference on Computer Human Interaction (CHI), pp. 1079-1088, April 2010, doi: 10.1145/1753326.1753486.

[7] A. K. Cebrowski and J. J. Garstka, "Network-centric warfare: Its origin and future," In US Naval Institute Proceedings, vol. 124, pp. 28-35, 1998.

[8] D. Caterinicchia and M. French, "Network-centric warfare: Not there yet," Federal Computing Week, vol. 9, 2003.

[9] J. Garstka, "Implementation of Network Centric Warfare," Transformation Trends, vol. 28, 2004.

[10] M. Stanovich, "Network-centric emergency response: The challenges of training for a new command and control paradigm," Journal of emergency management, vol. 4, no. 2 pp. 57-64, 2006.

[11] G. Hockey and J. Robert, "Operator functional state: the assessment and prediction of human performance degradation in complex tasks," Amsterdam, NL: IOS Press, vol. 355, pp. 1-383, April 2003.

[12] J-F. Gagnon, P. Jeuniaux, G. Dubé, and S. Tremblay, "Dynamic cognitive task modeling of complexity discovery A mix of process tracing and task analysis," Human Factors and Ergonomics Society Annual Meeting. vol. 55, pp. 1346-1350 September 2011, doi: 10.1177/1071181311551280.

[13] H. Ayaz et al., "Estimation of cognitive workload during simulated air traffic control using optical brain imaging sensors," Foundations of Augmented Cognition. Directing the Future of Adaptive Systems, pp. 549–558, January 2011, doi: http://dx.doi.org/10.1016/j.neuroimage.2011.06.023

[14] J-F. Gagnon, F. Couderc, M. Rivest, S. Banbury, and S. Tremblay, "Using SYnRGY to support design and validation studies of emergency management solutions," Proceedings of the 10th International ISCRAM Conference, May 2013.

[15] J. Allanson and S. H. Fairclough, "A research agenda for physiological computing," Interacting with computers, vol. 16, no. 5, pp. 857-878, October 2004.

[16] J. T. Coyne, C. Baldwin, A. Cole, C. Sibley, and D. M. Roberts, "Applying Real Time Physiological Measures of Cognitive Load to Improve Training," In D. D. Schmorrow, I. V. Estabrooke, & M. Grootjen (Eds.), Foundations of Augmented Cognition, Springer-Verlag Berlin Heidelberg, pp. 469–478, July 2009. doi:10.1007/978-3-642-02812-0.

[17] D. Lafond, H. Irandoust, S. Tremblay, W. Price, and A. R. Benaskeur, "A Context-sensitive functional model of teamwork operations," Proceedings of the 14th International Command and Control Technology Symposium. Santa Monica, CA, pp. 2-19, June 2009.

[18] S. Pappada, A. Geyer, K. Durkee, J. Freeman, and J. Cohn, "Modeling Operational Workload for Adaptive Aiding In Unmanned Aerial Systems (UAS) Operations," Aviation, Space, and Environmental Medicine, vol. 84, no. 4, pp. 331-332, April 2013.

 ISBN: 978-1-61208-341-4

HCI Dilemmas for Context-Aware Support in Intelligence Analysis

Daniel Lafond, René Proulx
Thales Research and Technology Canada
Thales Canada Inc.
Quebec City, Canada
e-mail: daniel.lafond@ca.thalesgroup.com
e-mail: rene.proulx@ca.thalesgroup.com

Alexis Morris, William Ross
Faculty of Computer Science
University of New Brunswick
Fredericton, Canada
e-mail: alexis.morris@unb.ca
e-mail: william.ross@unb.ca

Alexandre Bergeron-Guyard
Command, Control and Intelligence (C2I) Section
Defence Research and Development Canada – Valcartier
Quebec City, Canada
e-mail: Alexandre.BergeronGuyard@drdc-rddc.gc.ca

Mihaela Ulieru
School of Information Technology
Carleton University
Ottawa, Canada
e-mail: mihaela@theimpactinstitute.org

***Abstract*—The REcommending Cases based on cONtext (RECON) system is a prototype adaptive technology designed to support intelligence analysis using dynamic load balancing and advanced human-machine synergy. RECON combines a brain-computer interface, machine learning, and simulation in order to create an innovative case-based recommendation capability. Several dilemmas emerge when designing joint cognitive systems endowed with an adaptive capacity. Herein, we critically discuss these dilemmas related to human modeling and human-computer interaction.**

Keywords-adaptive system; human-computer interaction; context awareness; case-based recommendation; brain-computer interface; information relevance; modeling.

I. INTRODUCTION

Human-machine systems involve the often-complex interplay of human and technological components as interconnected actors sharing a common goal. These systems, while found in many domains, are particularly relevant in the case of defence and security, where intelligence analysts must make effective use of relevant information, communication, and logistic systems and technologies to improve situational awareness. Information overload is a critical area of concern for intelligence analysts who must sift through large volumes of data to uncover trends and make sense of unfolding situations [1].

The day-to-day activities of the intelligence analyst are driven by the intelligence cycle, illustrated in Figure 1. The intelligence cycle is defined as "*the process of developing raw information into finished intelligence for policymakers to use in decision-making and action*" [3]. The intelligence cycle encompasses many sensemaking tasks that the intelligence analyst must accomplish in an iterative fashion. Such tasks include: gathering relevant information; representing and organizing the information in a schematic way that will ease the analysis process; developing an understanding of the situation by subjecting the information to various hypotheses; and producing intelligence packages and recommendations for courses of action.

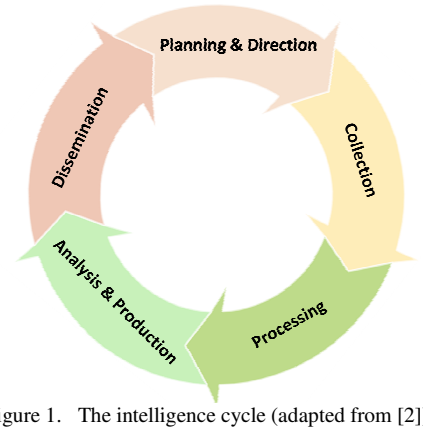


Figure 1. The intelligence cycle (adapted from [2])

As described by Pirolli and Card [4], the overall process is organized into two major loops of activities: (1) a *foraging loop* [5] that involves processes aimed at seeking, searching, filtering, reading and extracting information, possibly into some schema; and (2) a *sensemaking loop* [6] that involves iterative development of a mental model (a conceptualization) from the schema that best fits the evidence. This process is illustrated in Figure 2.

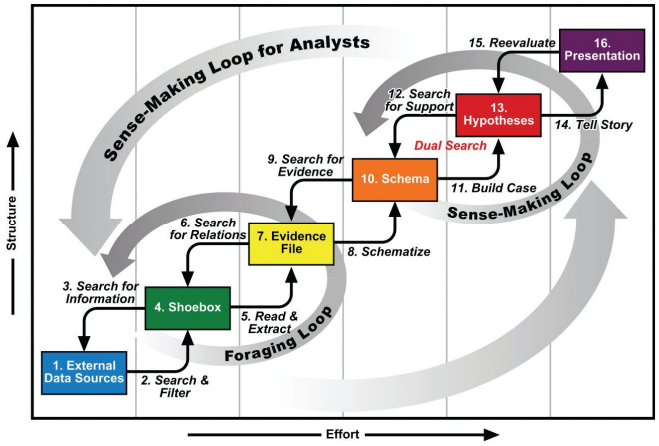


Figure 2. Notional model of sensemaking (from [4])

 ISBN: 978-1-61208-341-4

The analyst's activities within the intelligence cycle are subjected to a number of contextual factors (e.g., psycho-physiological and environmental) that can severely impede intelligence analysis due to excessive workload, time pressure, and uncertainty. The paper is organized as follows. Section II presents a prototype adaptive technology designed to support intelligence analysis using dynamic load balancing and advanced human-machine synergy. Section III discusses important dilemmas in the design of joint cognitive systems endowed with an adaptive capacity. Section IV concludes with a discussion of related work and directions for future work.

II. RECON: CONTEXT-AWARE CASE-BASED RECOMMENDATION FOR THE INTELLIGENCE VIRTUAL ANALYST CAPABILITY (IVAC)

The Intelligence Virtual Analyst Capability (iVAC) [7] is a recent Defence Research and Development Canada initiative that forms an intricate part of a Future Intelligence Analysis Capability (FIAC) [8]. iVAC is a knowledge system with an important human computer interface component that aims to alleviate the problem of cognitive overload by conducting a wide-variety of tasks. This initiative envisions a computerized software assistant supporting the intelligence analysts in sensemaking, while ultimately being capable of taking on autonomous analytical tasks in concert with other analysts (virtual or human).

As part of the research, an identification of iVAC sub-capability requirements was performed, based on literature reviews [9] and workshops held with experts from the military, the industry, and academia. The capabilities of the iVAC system were classified into seven broad categories:

- Context management;
- Acquisition of data, information, and knowledge;
- Activity monitoring, management, and evaluation;
- Learning of user and task models;
- Supporting complex intelligence tasks;
- Interaction with humans and other systems.

REcommending Cases based on cONtext (RECON) is a context-aware system being developed for integration with the iVAC. The central objective of RECON is to assist the intelligence analysts during the collection, processing, and analysis phases of the intelligence cycle (see Figure 1), by alleviating human-cognitive overload in two ways: firstly, by providing a system capable of sensing the user's contextual state using a brain-computer interface; and, secondly, by adapting the system to the user's context, identifying other similar contexts, and recommending relevant information to the user based on the system's level of awareness. The RECON architecture includes the following integrated layer components:

- *Brain-Computer Interface (BCI) layer:* Classifies user state and assesses user attention and interest to the displayed information;
- *Human-Computer Interaction (HCI) layer:* Presents adaptive interface elements and notifications;
- *Data layer:* Gathers information from multiple sources;
- *Context layer:* Transforms information from explicit and implicit sources into contextual knowledge;
- *Case-Based Recommendation (CBR) layer:* Provides case recommendations based on analyst's context.

The architecture components are conceptually organized according to the relations illustrated in Figure 3. A more thorough description of the RECON architecture can be found in [10]. The context management component of RECON is central to the adaptive system capability, combining HCI logs, data, and user-state classification from real-time analysis of electroencephalogram (EEG) signals to achieve contextual classification.

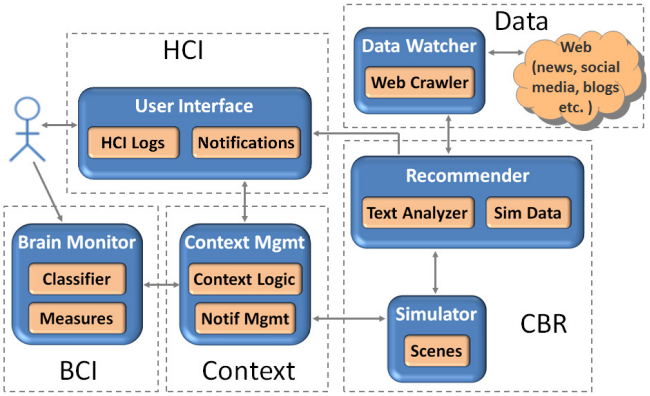


Figure 3. RECON components and relations

The system monitors the information being viewed by the user in real-time and assesses the user's degree of interest in regard to that information. This assessment aims to provide critical feedback to the case-based recommendation component, helping it provide more relevant recommendations to the user. Furthermore, EEG signal monitoring allows an assessment of the user's state in regard to the current pressures he/she is facing, which serves to modulate system behavior in accordance with this context (e.g., reducing user cognitive load through adaptive automation and postponing non-critical notifications).

State classification makes use of the Contextual Control Model [11], which posits that human decision makers can operate in one of four control modes:

- *Scrambled:* Planning is limited (or non-existent) and actions include trial-and-error, reactive or random approaches with no forward thinking;
- *Opportunistic:* Planning is limited and actions are based on salient situation characteristics;
- *Tactical:* Planning is present but restricted to the current situation and actions are guided by procedural or rule-based decision making;
- *Strategic:* Planning extends beyond the current situation and actions consider high-level goals and global context.

The selection of a control mode is a function of the subjective estimation of the time required to perform a task and of the time available [12]. While the human analyst can

 ISBN: 978-1-61208-341-4

dynamically adapt his or her control mode to cope with situational constraints, RECON aims to recognize these changes in control mode and adjust its behavior accordingly. One major technical and scientific challenge is to derive effective and reliable classification models using EEG signals as inputs [13] in the applied context of intelligence analysis. A two-stage process is employed to achieve this. First, an experimental training set will provide the critical human data necessary for initial model comparison and selection. Secondly, individual user feedback will be incorporated to allow validation and fine-tuning of the classification rules for each analyst. Together with the integrated system components shown in Figure 3, these will allow RECON to achieve its goal of context-aware, cased-based recommendation for iVAC.

III. DILEMMAS

Five key dilemmas, relevant to the design of adaptive systems at large are critically discussed below. These generic dilemmas are especially relevant to human modeling (model selection and calibration) and human-computer interaction (model transparency, user feedback, and explicit vs. implicit contextual inputs).

A. Model Selection: Statistics vs. Machine Learning

A first dilemma for modeling user state is whether to opt for statistical analyses based on the General Linear Model (GLM) or for a Machine Learning (ML) algorithm to appropriately capture the underlying pattern of cerebral activity associated with a given state. The GLM approach traditionally taught to neuroscientists has a proven track record and comes with robust analysis software [14], yet the linearity constraint means that complex non-linear relations cannot be ”discovered” using this method (i.e., the underfitting problem) [15]. On the other hand, the linearity contraint makes the GLM very robust to noise (i.e., measurement error or intrusions from confounding factors), thus minimising the overfitting problem. Underfitting means that the model lacks functional flexibility to capture a phenonemon, while overfitting means that the model’s flexibility allows it to “fit” both the true regularities in the data but also false patterns that are actually noise (leading to an overestimation of a model’s real accuracy) [16]. ML algorithms (or “data mining” algorithms) provide highly flexible models capable of discovering highly complex patterns in datasets. However, the flexibility of ML algorithms makes them vulnerable to overfitting.

To resolve this dilemma, the approach proposed here is to concurrently consider models that differ in their functional flexibility and compare their predictive accuracy [17][18]. Indeed, the gold standard in model selection is to assess a model’s predictive accuracy by using one (or several) “training samples” for model calibration (i.e., to learn the pattern in the data) and one (or more) “test samples” for model validation. Models that tend to overfit to noise in the data will thus tend to perform worse on the test sample than on the training sample (i.e., a phenomenon called shrinkage) [19]. Alternatively, models that start simple and “grow” to accommodate more complex patterns in the data (e.g., decision trees and cascade correlation) can include stopping rules that check when the prediction error stops improving (i.e., finding the “sweet spot” between underfitting and overfitting).

B. Individual Calibration vs Collective Calibration

A second dilemma relevant to user-state modeling is whether to perform model calibration at the group level (i.e., resulting in a single model for all potential users) or at the level of the individual. Clearly, individual modeling has the disadvantage of requiring a new data collection for each user in order to extract an individualized model. Nonetheless, this individualized approach may be necessary in order to reach high levels of model accuracy, particularly when the average is the result of idiosyncratic patterns [20][21]. The alternative is to treat individual differences as noise (leading to a potential underfitting of the user state).

The solution proposed herein is to focus on discriminating between broad state categories (as opposed to continuous scales of the concept of interest), which may not require individual user modeling to achieve a satisfactory accuracy. For example, RECON could use a classification model, such as low, medium, and high, to discriminate among different categories of “interest toward a type of information,” instead of using a continuous equal-interval scale.

C. Model Transparency to the User

A third dilemma, related to human-computer interaction, is whether or not to display to the user the model’s inputs, its logic, and its resulting assessment. A transparent model offers the possibility to increase user trust, but there is also the risk of a backlash if the user disagrees with the model or simply does not understand it. Conversely, a “black box” model may foster doubt and mistrust in the system. This issue also relates to the classic invisibility dilemma which involves choosing between minimizing distractions from the primary task and providing added value through explicit interaction [22].

The proposed solution to this dilemma is to make only the model output (e.g., the inferred state) transparent to the user, thus reducing risks and distractions yet allowing the user to develop a sense of trust over time as a function of the tool’s classification accuracy. For example, RECON could show the analyst the currently estimated control mode (e.g., scrambled, opportunistic, tactical, or strategic) without displaying the current input values and the classification model.

D. Learning Model Based on User Feedback

A fourth dilemma involves whether or not to collect user feedback in order to sample the correct state at different moments in time, at least for an initial model calibration phase. The alternative is to resort to indirect indicators of user state such as observer judgments or behavior patterns associated with each state (note that unsupervised learning methods are not considered here) [23].

The proposed solution to this dilemma is to use both approaches in order to combine self-ratings and observer

 ISBN: 978-1-61208-341-4

ratings into a more reliable metric, with observers being supported by access to behavioral markers to help discriminate between the different user states considered. For example, the classification of the control mode in RECON could be calibrated based on feedback in a training phase, using self reports (after the fact) from the intelligence analysts' perceived control mode at differents moments in time, combined with judgments from an expert observer.

E. Explicit vs Implicit Contextual Inputs

A fifth dilemma involves knowledge about user context, which is central to system adaptation. Context is what describes the environment, situation, state, surroundings, tasks, social settings, and roles, among other things [10]. This context evolves according to events and changes occurring during system operation either by direct explicit interactions from the user (e.g., a user manually indicates current context parameters such as time pressure, psycho-physiological state, availability, and current interest in certain types of information) or indirect implicit interactions based on the situational context (e.g., automatic data monitoring, HCI monitoring, and sensor-based perception).

Explicitly specifying context affords the user a sense of control over the system and provides contextual data that may not be otherwise available. However, a system that relies too much on explicit context will put a heavier workload on the user as he or she must provide a larger amount of information to the system, requiring a more complex graphical-user interface and a larger number of manipulations which may interfere with the user's ability to focus on the task at hand. Conversely, a system that emphasizes implicit context frees the user from tedious data input operations, but requires the system to monitor data and perform reasoning to infer contextual information. This requires a significant a priori effort to develop effective user-state and contextual classification models.

The proposed solution to this dilemma is to combine both explicit and implicit context within RECON. Implicitly, context will be derived from the BCI, HCI, and Data layers (Figure 3), while other contextual information such as a user's current task will be obtained through explicit user input.

IV. CONCLUSION

The RECON system, currently in development, aims at providing an innovative context-aware case-based recommendation framework for the intelligence virtual analyst capability (iVAC). This work builds on previous research in intelligence analysis [7][8], context-aware systems [9], BCI [13], human factors [11][12] and classification modeling [15][20]. It is expected that in situations involving information overload, uncertainty, and time pressure, the effectiveness of intelligence analysts can be significantly improved through context-aware adaptive systems. The approach described in this paper relies heavily on psycho-physiological measurement to infer the user's cognitive state in order to implicitly coordinate the system and the user. An alternative approach is to focus on explicit coordination through human-machine teamwork, enabled through interaction with a virtual assistant [7]. The iVAC initiative seeks to combine these two complementary approaches.

This paper presents five HCI dilemmas for context-aware support in intelligence analysis related to model selection, calibration, model transparency, user feedback, and contextual inputs. Moreover, how these are addressed in RECON is also presented, along with the architecture and core motivations. While the five HCI dilemmas delimit a solution space for designing adaptive joint cognitive systems, the existence of a general optimal configuration is unlikely. The solutions proposed in the context of RECON may not provide an ideal cost-benefit tradeoff in other contexts (and this may also depend on the user). A future design methodology that could parse various combinations and determine the optimal configuration for a given context/user would be very useful. There are also interdependencies between these dilemmas that need to be better understood. Finally, it should be noted that this non-exhaustive list of dilemmas relevant to adaptive systems could be complemented by additional HCI dilemmas such as those identified for supervisory control tasks [24].

With its focus on adaptive off-loading and high-relevance system recommendations, RECON aims to advance the state of the art in the study of context-management systems, case-based recommendation, brain-computer interfaces, and human-computer interaction, through an upcoming proof-of-concept experiment.

ACKNOWLEDGMENT

Thanks are due to Prof. Amedeo D'Angiulli and Prof. Michael Fleming for their insights and institutional support. This work was funded by Defence R&D Canada, by Thales Research and Technology Canada, and by a research partnership grant from the Department of National Defence of Canada and the Natural Sciences and Engineering Research Council of Canada to Prof. D'Angiulli.

REFERENCES

[1] E. S. Patterson et al., "Aiding the intelligence analyst in situations of data overload: From problem definition to design concept exploration," Institute for Ergonomics/Cognitive Systems Engineering, ERGO-CSEL 01-TR-01, 2001.

[2] M. Chesbro, "Intel-Cyclopedia: A Guide to Sources of Information for the Intelligence Community," Homeland Security Digital Library. http://www.hsdl.org/ [retrieved: April 2014].

[3] Central Intelligence Agency, "The work of a Nation," Library of Congress, 2009.

[4] P. Pirolli and S. Card, "The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis," Proc. IEEE Symp. Computational Intelligence Analysis, May 2005, pp. 1-6.

[5] P. Pirolli and S. K. Card, "Information foraging," Psychological Review, 106, pp. 643-675, 1999.

[6] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card, "The cost structure of sensemaking, " Paper presented at the INTERCHI '93 Conference on Human Factors in Computing Systems, Amsterdam, Apr. 1993, pp. 1-9.

 ISBN: 978-1-61208-341-4

[7] D. Gouin, V. Lavigne, and A. Bergeron-Guyard, “Human-computer interaction with an intelligence virtual analyst,” in Proc. Knowledge Systems for Coalition Operations, Pensacola, FL, Feb. 2012, pp. 1-5.

[8] D. Poussart, “Future intelligence analysis capability—towards a cohesive R&D program definition,” DRDC Valcartier, TM 2012-9999, 2012.

[9] J. Hong, E. Suh, and S. J. Kim, “Context-aware systems: A literature review and classification,” Expert Systems with Applications, vol. 36, no. 4, pp. 8509-8522, 2009.

[10] W. Ross, A. Morris, M. Ulieru, and A. Bergeron-Guyard, “RECON: An Adaptive Human-Machine System for Supporting Intelligence Analysis,” IEEE International Conference on Systems, Man, and Cybernetics, Oct. 2013. pp. 782-787, doi: 10.1109/SMC.2013.138.

[11] E. Hollnagel and D. D. Woods, “Joint cognitive systems: Foundations of cognitive systems engineering.” Boca Raton, FL: Taylor and Francis, 2005.

[12] M.-E. Jobidon, R. Rousseau, and R. Breton, “Time in the Control of a Dynamic Environment,” Proc. of the Human Factors and Ergonomics Society 48th Annual Meeting (pp. 557-561), Sept. 2004, doi: 10.1177/154193120404800360.

[13] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi, “A review of classification algorithms for EEG-based brain–computer interfaces.” Journal of neural engineering, vol. 4, March 2007, pp. R1-R13.

[14] G. D. Hutcheson and N. Sofroniou, “The multivariate social scientist: Introductory statistics using generalized linear models,” Sage, 1999.

[15] M. A. Pitt, W. Kim, and I. J. Myung, “Flexibility versus generalizability in model selection,” Psychonomic Bulletin & Review, 10, pp. 29-44, March 2003.

[16] S. Roberts and H. Pashler, “How persuasive is a good fit? A comment on theory testing. Psychological Review, vol 107, April 2000, pp. 358–367, doi: 10.1037/0033-295X.107.2.358.

[17] M. Browne, “Cross-validation methods,” Journal of Mathematical Psychology, vol 44, March 2000, pp. 108–132.

[18] J. R. Busemeyer and Y. Wang, “Model comparisons and model selections based on the generalization criterion methodology,” Journal of Mathematical Psychology, vol 44, March 2000, pp. 171–189.

[19] B. S. Everitt. Cambridge Dictionary of Statistics (2nd Edition), CUP, 2002.

[20] W. K., Estes and W. T. Maddox, “Risks of drawing inferences about cognitive processes from model fits to individual versus average performance,” Psychonomic Bulletin & Review, vol 12, June 2005, pp. 403–408.

[21] P. N. Mohr and I. E. Nagel, “Variability in brain activity as an individual difference measure in neuroscience?” Journal of Neuroscience, vol 30, June 2010, pp. 7755-7757; doi: 10.1523/JNEUROSCI.1560-10.2010.

[22] A. Schmidt, M. Kranz, and Paul Holleis. “Interacting with the ubiquitous computer: towards embedding interaction,” In Proceedings of the joint conference on Smart objects and ambient intelligence, October 2005, pp. 147–152.

[23] S. Asteriadis, P. Tzouveli, K. Karpouzis, and S. Kollias. "Estimation of behavioral user state based on eye gaze and head pose—application in an e-learning environment." Multimedia Tools and Applications, vol 41, Feb. 2009, pp. 469-493.

[24] T. B. Sheridan, “HCI in supervisory control: Twelve dilemmas,” in Human Error and System Design and Management, Elzer, P., Kluwe, R. & Boussoffara, B. (Eds.), Springer-Verlag: London, pp. 1-12, 2000.

 ISBN: 978-1-61208-341-4

Driving Style Recognition for Co-operative Driving: A Survey

Anastasia Bolovinou, Angelos Amditis
I-SENSE group
Institute of Communications and Computer Systems
Athens, Greece
abolov@iccs.gr;a.amditis@iccs.gr

Francesco Bellotti
Dept.of Naval, Electric, Electronic and Telecommunications Engineering, University of Genoa
Genoa, Italy
franz@elios.unige.it

Mikko Tarkiainen
VTT
Tampere, Finland
Mikko.Tarkiainen@vtt.fi

***Abstract*—This paper serves as a critical survey for automatic driving style recognition approaches and presents "work in progress" ideas that can be used for the development of intelligent context-adaptive driving assistance applications. Furthermore, a preliminary specification of a context-adaptive application that can be described by the following three steps is provided: at first, driving style is automatically classified into one out of a set of predefined classes that are learnt through historic driving and trip data; secondly, based on the driving style recognition a context-adaptive driving application is proposed; thirdly, eco-safe and co-operative driving behaviour can be rewarded by the system by introducing a serious game theoretic approach. While the focus of this paper lies on reviewing the state of the art for implementing the first step, providing the high-level specification of the two other steps offers valuable insight on the requirements of such collaborative driving application.**

Keywords- driving behaviour; vehicle dynamics; time-series analysis; supervised learning; classification; co-operative system.

I. INTRODUCTION

Driving is in essence a multi-factor cognition task that can only be perceived in the context of underlying road layout, traffic, weather and social behaviour conditions in a framework where the action of an individual driver is affected by the actions of other drivers and travellers that co-exist temporarily: the driver recognizes the road environment including road layout, traffic conditions, and the behaviour of nearby vehicles, e.g., distance from the vehicle in front, and decides actions to take, such as accelerating, braking, and/or steering. With experience gained each driver develops an individual behaviour behind the wheel, which could impact safety, fuel economy, and road congestion, among other things. For example, a driver, usually, maintains a comfortable time gap to the leading vehicle by adjusting his/her own threshold based on traffic conditions.

Being able to dynamically recognize the driving style is invaluable information for modern Intelligent Transport Systems and Road Operators. More specifically, being able to collect contextual information about the driving style coupled with specific traffic or road characteristics, allows the road operator to perform reasoning about safety characteristics of road usage and react upon that information: for example, Zhang et al. [1] assess vehicle dynamic information and perform a categorization of drivers' acceleration patterns in specific road curves. If this information is compared with speed limit information and accident statistics in the specific road segment under investigation, specific adjustments of the road infrastructure can be proposed in order to minimize the possibility of bad driving behaviour.

Recognizing the driving style is also important in any effective Advanced Driver Assistance System (ADAS) that aims to increase its acceptability by being adaptive to the driver's behaviour: One good example is the Adaptive Cruise Control (ACC) system presented in [1], which adapts its output before entering a curve based on road context data. In contrast with a conventional ACC system that simply maintains the host vehicle speed at a set value if there is no preceding car regardless of the road conditions, the proposed system can provide individual drivers with customized speeds based on their preferred speeds, deceleration rates, and lateral acceleration. Such an adaptive system is expected to increase ride comfort and highly decrease the need for driver intervention in the ACC functionality. More broadly, future ADAS are expected to rely heavily on driving intention recognition and driving behaviour prediction, in order to choose a suitable control strategy to assist and/or warn the driver or even intervene in an automatic manner [2].

Within a collaborative mobility concept, the performance of future ADAS applications can be optimized based on automatic recognition of driving behaviour: for example, the suggested headway, from the vehicle in front, in a co-operative ACC application, can be adapted to the dynamically changing driving style of the driver of the vehicle in front. In a broader consideration, categorizing the driving style of the driver , e.g. as "safe driver", "aggressive driver" or "good fuel economy driver", could be used to encourage community-friendly driving styles; combined with driver records and models of ideal driving style, one's

 ISBN: 978-1-61208-341-4

driving style could be compared and used, as an immediate feedback to the driver while driving [3]. In a community building serious game approach, driver coaching and rewarding good driving habits could help promote ecological and safer driving.

In this work, recent approaches that try to automatically identify driving behaviour by recognizing specific car-following and pedal/steering wheel operation patterns will be reviewed in order to identify suitable methods for context-adaptive and collaborative driving applications as these will be investigated within the European TEAM project [4]. Note that approaches that deal with vision-based driver state monitoring (by tracking the driver's face or body motions, see the review of Kang [5]) are not studied here as we want to focus on an inertial base system that enables a discrete, unobtrusive, and seamless recognition of the driver's behaviour.

The structure of this paper is as follows. In Section II, a state of the art review is provided that concludes with observations on the features that appear in the methods under review. In Section III, the functional architecture of a future co-operative driving application that can adapt to the recognized driving style and upon which, a co-operative game-theoretic approach can be built, is presented. Conclusions and ideas for future work are included in Section IV that concludes this work.

II. METHODS FOR AUTOMATIC DRIVING STYLE RECOGNITION IN THE LITERATURE

A. Related Work

Benavente et al. [6] study three different datasets to examine the relationship between aggressive driving and roadway characteristics, such as type of road, speed limit, number of travel lanes and presence of curbs. Aggressive driving behaviours include speeding, failure to stop, lane violations (such as improper passing), and severe violations (such as operating recklessly). Based on an empirical study on Automatic Cruise Control use by 118 subjects, Fancher et al. [7] classified drivers into five categories: flow conformist, extremist, hunter/tailgater, planner, and ultraconservative. Depending on their velocity and distance from the vehicle in front, a simple empirical model, which divided this two-dimensional feature space into classes of interest by applying rule-based thresholds, was applied. Since such large-scale testing with many different drivers are difficult to be performed while data annotation is too time-consuming and on the other hand real driving styles can vary a lot depending on the country's driving culture and road/weather condition, machine learning methods for discovering patterns and classifying driving styles based on them is highly preferable.

Taking advantage of the recent advancement in machine learning and data mining algorithms, big multi-dimensional time-series data can be explored in order to discover repeating patterns and spatio-temporal relationships among them [8]. Moreover, based on the software/hardware advances in communications and automotive on-board diagnostics units, we are able to record the high-frequency real-time driving information. In this review we are interested in works that discover patterns from rich driving data, which include vehicle signals captured in the context of a specific trip (terrain, weather, traffic information may be included).

In the work of Mudgal et al. [9], speed profiles of different drivers at a roundabout have been modelled, with average circulating speed and non-linear parameter such as position of maximum acceleration determined using Bayesian inference methods. In addition, vehicular emissions were estimated using past experimental data. It is found that speed profiles differ significantly across drivers, as do the mean speeds at the circulating path of the roundabout. The model provides a second-by-second speed profile that can be used for deriving acceleration profile, which can be used for emission hotspots or aggressive driving behaviour recognition. The average circulating speed can also be used as a parameter for developing driving cycles for corridors that include roundabouts. In the work of Spiegel et al. [10], a Singular Value Decomposition bottom-up algorithm that identifies, internally homogeneous, time series segments is adopted. To recognize recurring patterns, the established time series segments were grouped via agglomerative hierarchical clustering. Subsequently, recurring sequences of grouped segments, which can be considered as classes of high-level driving context, can be retrieved.

Spectral analysis of velocity, following distance (only simulation data), gas and brake pedal signals is used for signal representation by C. Miyajima et al. Then, multiple component Gaussian Mixture Models (GMM), applied on a 0.32-s frame length, are used for data modelling. The model fits a GMM for each driver and performs driver identification among a small set of subjects. Targeting at the same driver identification objective, Ly et al. [12] apply automatic extraction and classifications of three simple driving events, defined as brake, acceleration and turn event (GPS positioning data are ignored). Support Vector Machine (SVM) and K-means clustering are compared for a 2-class classification (driver A and driver B), while the classification performance does not exceed 65%.

Johnson and Trivedi [13] detected and classified driving manoeuvres using a smartphone's accelerometer and gyro sensors mounted in the car. In a similar approach, Sathyanarayana et al. applied SVM classification in order to compare the automatic driving maneuver recognition that is based on signals from smart phones against using the CAN signals from the vehicles and equal performance of the two methods is reported.

Amata et al. [15] introduce two prediction models for driving events' recognition: the first is based on multiple linear regression analysis which predicts whether the driver will steer or ease up on the accelerator, or brake; the second predicts driver decelerating intentions using a Bayesian Network. The proposed models predict the three driving actions with over 70% accuracy when the use cases are split into 9 categories of intersection classes. Kishimoto and Oguri [16] also proposed a prediction method that forms an Autoregressive switching-Markov Model (AR-HMM) in

 ISBN: 978-1-61208-341-4

order to predict stop probability focusing on a certain period of past movements.

Although not related only with driving style, similar works appear in energy consumption prediction for electric vehicles. In a review of driving behaviour recognition methods for fuel efficiency in hybrid vehicles, Wang and Lukic [17] divide driving styles into three categories: mild drivers (calm driving or economical driving style), normal drivers (medium driving style) and aggressive drivers (sporting driving style). A spine regression model for predicting gasoline consumption rate from speed, acceleration and heading degree information is applied on real driving data (a box-cox transformation to the response variable gives improved modelling) by Nie et al. [18]. Similarly, Quek and Ng [19], train SVM and multinomial logistic regression models but the model is evaluated only for a small set of categories. A last example from the electric vehicles field is the work of Ferreira et al. [20], where a Naïve Bayes classifier was used to classify driver behaviour in several pre-defined classes with respect to electric energy consumption (classes are defined based on percentage of the *state of charge* level decreasing from the ideal driver). As input data, discretized weather information (temperature and raining information), average speed, traffic information, road type, EV age and type, drive mode (work or leisure) and drive period (morning, afternoon, night) were utilized but no evaluation of the method is presented.

Although the majority of the works presented deal with a classification problem, i.e. [9-14][16][20], formulating the problem as a regression problem i.e. [15][18][19], is considered very useful as it gives valuable insights in the recognition problem in hand, by providing estimation of each factor contribution to the event that we wish to recognize.

In all the previous works and independently of the underlying data model (GMMs, Bayesian inference such as HMM) or its absence (SVMs, SVD), a significant correlation of the low-level data being observed and the classes that represent the driver behaviour is assumed. This might be true for classes that represent low-level information such as speed, acceleration or stopping profile, as in [9][10][16], or even some specific manoeuvre detection as in [13 - 15]. However, when higher level behaviour recognition is required, such as driver intention recognition, the existence of contextual information related to the surrounding conditions needs to be taken into account despite the absence of relevant clues. To overcome this problem, Taniguchi et al. [21] assume that contextual information has a double articulation statistical structure. The underlying assumption is that since a concrete value of driving behaviour cannot be easily predicted, an alternative task can be to predict contextual information, i.e., hidden states of these probabilistic models. Following the above line, Fox et al. [22] use an extension of the Hierarchical Dirichlet Process Hidden Markov Model (HDP-HMM), appropriate for dynamic time series modelling, called sticky HDP-HMM. By using this model, the analyser can estimate segments and obtain sequences of hidden state labels (letters) without fixing the number of hidden states. Similarly, in the work of He et al. [23], a double-layer Hidden Markov Model (HMM) is developed for driving intention recognition and behaviour prediction using manoeuvring signals and vehicle state measured by a driving simulator. Each multi-dimensional Gaussian HMM bank in the lower layer corresponds to nine short- braking/acceleration manoeuvres and three steering driving behaviour HMMs while upper-layer multi-dimensional discrete HMMs are built for long-term driving intention in a combined working case. Finally, a semi-supervised time-syntactic pattern recognition approach (by discretizing the values of on-board sensors into simple brake/steering events) is applied for learning models of the driving behaviour of truck drivers by Verwer et al. in [24].

B. Notes on Feature Representation Appopriate for Automatic Driving Style Recognition

Although it is difficult to directly compare the methods presented due to lack of shared common datasets, one element that differentiates this set of methods is the feature representation used. Since this a factor that may judge the power of the overall method, in the followings, we proceed with some observations and we draw some directions for future work in the field.

In terms of driving cycles characteristics, the research has started since as early as 1978 when Kuhler and Karstens [25] introduced 10 aggregate driving behaviour parameters: average speed, average speed excluding stop, average acceleration, average deceleration, mean length of a driving period, average number of acceleration deceleration changes within one driving period, proportion of standstill time, proportion of acceleration time, proportion of deceleration time, and proportion of time at constant speed. More recently, Huang [26] study the influence of 11 parameters on driving cycle recognition and argue that by using 4 of them the prediction result is satisfactory. Higher level features for driving style recognition could be provided if a feedback loop with an advanced driver assistance application is established so that for instance the reaction time from the changing of the signal light to the actual movement of the car is measured, as proposed in [19] and also envisioned in TEAM applications [4].

Feature selection and their representation is critical in the applicability and the discriminative power that a machine learning algorithm can demonstrate for a specific problem in hand. Since the extraction of contextual information for dynamic events like driving requires high computational demanding statistical models (like the ones used in [21-24]) which may not be easily adapted for real-time systems, the robust representation of low-level data is considered necessary since:

- As noted by Liu [27], instantaneous accelerator or brake pedal positions are very noisy signals in the sense that the moment-to-moment position of the pedals do not reveal the actual vehicle speed (due to vehicle dynamics history).
- There is noise in vehicle sensors' measurement and dynamics estimation (even if filtering is applied, e.g., Kalman filter).

While most of the works presented in Section II.A, deal directly with time-series data, without performing a

 ISBN: 978-1-61208-341-4

quantization of the signal, there are also some works that choose to work on a histogram-based representation of aggregated data. For example in the work of Ly et al. [12], "histogramming" of the extracted time series vector into 5 bins seems to help: "Using 5 bins histogram reduces the feature vectors size and helps alleviate over fitting when learning with limited data size. Typical signal statistics such as the min, max, mean, and variance are included in the feature vector. Additionally, the duration of the event is also included". The feature vectors for turning activity recognition include the histograms of both the angular velocity and the longitudinal acceleration. An acceleration/deceleration histogram by fitting a 3rd order polynomial curve to the speed data only at the regions where significant acceleration / deceleration was observed is obtained by Quek and Ng [19].

The advantage of this latter approach is that they can proliferate by the big progress in histogram-based feature vector processing in the fields of text (initially) and image processing where impressive results have been obtained by using a dimensionality reduction technique known as bag of features [28]. A different interesting approach, that handles time-series driving data as a two-dimensional grey-scale image data, has been presented by Griesche in [29]. Converting time-series data into histograms has also been proposed recently by Lin and Li [30] with promising results.

III. A FUTURE APPLICATION: COLLABORATIVE-DRIVING APPLICATION THROUGH GAMING

Gamification can be a solution to the challenge of long term involvement of drivers in cooperative driving. In the agent-based approach of Rossetti et al. [31], the authors focus on driving behaviour elicitation by promoting synergies in a simulated artificial society on a participative basis. In the recently started TEAM integrated European project [32], the goal of the next generation cloud enabled co-operative elastic mobility is pursuit through the development of adaptive transport and driving applications used by a community of users. Part of the preliminary specification work on a Serious Gaming and Community building application which requires a driving style classification component is presented hereafter.

In TEAM, a cloud-based architecture is assumed where intelligent algorithmic components, which are considered as enablers for TEAM applications, run on a distributed cloud server system that also stores local dynamic map information and drivers/travellers history of trips. Moreover, a social network management enabler stores information that the users are willing to share with their co-commuters through smartphones. As a mean of involving the user in a collaborative driving task in order to achieve more fluid and ecological behaviour the TEAM Serious Game and Community Building (SG-CB) application is specified. SG-CB vision is to implement a travelling game based on participation of a community of users where the system provide drivers with "virtual community currency" related with eco-safe driving behaviour (e.g., driver gets a virtual coin if he/she keeps correct headway, he/she can spend the virtual coin later).

The game targets in essence, in coaching the driver towards obtaining a better community performance. For this reason, the SG-CB application should be able to recognize typical driving behaviours such as: headway profile, stopping profile in intersections, longitudinal and lateral acceleration profile. Therefore, a multi-class classification of the driving style that is dynamically updated is proposed as an enabler called "Driving Style Classifier".

As shown in Figure 1, driving style classification is expected to run online and its output will be assessed by the SG-CB application in order to present to the driver, through HMI, messages relative with the game objectives. The inputs of this component are based on vehicle and context data available from the cloud (like traffic conditions or terrain characteristics based on geo-position). For recognizing different driving styles, several driving indicators are envisioned to be defined using mass past driving data and the classifier will be responsible for assigning a weight corresponding to the similarity of the real time driving event with each of these available indicators.

An appropriate dynamic data representation selected based on the directions derived in section II.B and a robust machine learning algorithm like SVM will be the internal components of the classifier component. For training the classifier, the project plans to create driving reference databases by integrating driving data for safety analysis by previous TeleFOT [33], in European level, and national Tele-ISA [34] and Trafisafe [35] field trials. Using aggregates over these data, driving indicators reference database for highway and urban scenarios will be built. Examples of such indicators are the percentage of kilometres driven at more than 10 km/h over the speed limit, the number of hard longitudinal and lateral accelerations/decelerations, high yaw rate angles, hard braking events per 100km, high acceleration/deceleration combined with rain or darkness conditions per 100km.

IV. CONCLUSIONS AND FUTURE WORK

Requirements and methods for automatic driving style recognition from vehicle and trip data have been studied. Emphasis has been given in the feature selection strategy and a novel promising method for bag of patterns classification, based on time series data turned to histogram-based features, has been identified by combining clues from the literature and the recent advances in the European automotive research projects. Future implementation of a driving style classification module in the terms of a gaming collaborative driving application was drafted based on three categories of driving style: safe, eco-friendly and fluid-promoting in the ACC context.

ACKNOWLEDGMENT

This work was also supported by the European Commission under TEAM integrated project (FP7-ICT-2011-8). The authors would like to thank all partners within TEAM for their cooperation and valuable contribution.

 ISBN: 978-1-61208-341-4

REFERENCES

[1] D. Zhang, Q. Xiao, J,Wang and K. Li, "Driver curve speed model and its application to ACC," International Journal of Automotive Technology, 2013, vol. 14, no. 2, pp. 241−247.

[2] Mauro Da Lio, Francesco Biral, Marco Galvani and Andrea Saroldi, "Will Intelligent Vehicles Evolve into Human-peer Robots?," 2012 IEEE Intelligent Vehicles Symposium, Conference proceedings, pp. 304 – 309.

[3] S. Rass, S. Fuchs, and K. Kyamakya, "A Game-Theoretic Approach to Co-operative Context-Aware Driving with Partially Random Behavior, Smart Sensing and Context," Lecture Notes in Computer Science, vol. 5279, 2008, pp. 154-167.

[4] A. Amditis, P. Lytrivis, I. Karaseitanidis, M. Prandtstädter, and I. Radusch, "Tomorrow´s Transport Infrastructure: from Static to Elastic Mobility," Proc. of the 20th ITS World Congress 2013, Tokyo, 14-18 October 2013.

[5] H. Kang, "Various Approaches for Driver and Driving Behavior Monitoring: A Review," Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on, pp. 616-623, Dec. 2013, doi: 10.1109/ICCVW.2013.85.

[6] M. Benavente, M. A. Knodler, and H. Rothenberg, "Analysis of the Relationship between Aggressive Driving and Roadway Characteristics Using Linked Data," Institute of Transportation Engineers Annual Meeting and Exhibition, 2007, Pittsburgh, PA.

[7] P. Fancher et al., "Intelligent Cruise Control Field Operational Test," Technical report, vol. I, U.S. Dept. of Transportation NHTSA, 1998.

[8] J. Lin, S. Williamson K. Borne, and D. DeBarr, "Pattern Recognition in Time Series," book chapter in Advances in Machine Learning and Data Mining for Astronomy", Eds. Kamal, A., Srivastava, A., Way, M., and Scargle, J. Chapman & Hall, Mar 2012.

[9] A. Mudgal, S. Hallmark, A. Carriquiry, and K. Gkritza, "Driving behavior at a roundabout: A hierarchical Bayesian regression analysis," Transportation Research Part D: Transport and Environment, vol. 26, January 2014, pp. 20-26, ISSN 1361-9209, doi: 10.1016/j.trd.2013.10.003.

[10] S. Spiegel, J. Gaebler, A. Lommatzsch, E. De Luca, and S. Albayrak, "Pattern recognition and classification for multivariate time series," Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data (SensorKDD '11). ACM, NY, USA, pp. 34-42.

[11] C. Miyajima et al., "Driver Modeling Based on Driving Behavior and Its Evaluation in Driver Identification," Proceedings of the IEEE , Feb. 2007, vol. 95, no. 2, pp.427-437, doi: 10.1109/JPROC.2006.888405.

[12] M. V. Ly, S. Martin, and M. M. Trivedi, "Driver Classification and Driving Style Recognition using Inertial Sensors," IEEE IV2013, June 23-26, 2013, pp. 1040-1045, Gold Coast, Australia.

[13] D. A. Johnson and M. M. Trivedi, "Driving Style Recognition Using a Smartphone as a Sensor Platform," 14th International IEEE Conference on Intelligent Transportation Systems, October 5-7, 2011, pp. 1609-1615, Washington, DC, USA.

[14] A. Sathyanarayana, S. O. Sadjadi, and J. H. Hansen., "Leveraging sensor information from portable devices towards automatic driving maneuver recognition," IEEE 15th International Conference on Intelligent Transportation Systems (ITSC), 2012, pp. 660–665.

[15] H. Amata, C. Miyajima, T. Nishino, N. Kitaoka, and K. Takeda, "Prediction model of driving behavior based on traffic conditions and driver types," IEEE 12th International Conference on Intelligent Transportation Systems, 4-7 Oct. 2009, pp. 1-6.

[16] Y. Kishimoto and K. Oguri, "A Modeling Method for Predicting Driving Behavior Concerning with Driver's Past Movements," Proc. IEEE International Conference in Vehicular Electronics and Safety, Sept. 2008, pp. 132 – 136, doi: 10.1109/ICVES.2008.4640888.

[17] R. Wang and S.M. Lukic, "Review of driving conditions prediction and driving style recognition based control algorithms for hybrid electric vehicles," IEEE Vehicle Power and Propulsion Conference (VPPC) , 6-9 Sept. 2011, pp. 1-7.

[18] K. Nie, L. Wu, and J. Yu, "Driving Behavior Improvement and Driver Recognition Based on Real-Time Driving Information," technical report in CS229 Project, Stanford university, 2013.

[19] Z. F. Quek and E. Ng, "Driver Identification by Driving Style," technical report in CS 229 Project, Stanford university 2013.

[20] J. Ferreira, V. Monteiro, and J. L. Afonso, "Data Mining Approach for Range Prediction of Electric Vehicle," Conference on Future Automotive Technology - Focus Electromobility, 26-27 March 2012, Munich, Germany, pp. 1-15.

[21] T. Taniguchi, S. Nagasaka, K. Hitomi, N. P. Chandrasiri, and T. Bando, "Semiotic Prediction of Driving Behavior using Unsupervised Double Articulation Analyzer," 2012 Intelligent Vehicles Symposium, Alcalá de Henares, Spain, June 3-7, 2012, pp. 849 - 854, doi: 10.1109/IVS.2012.6232243.

[22] E. B. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky, "The sticky hdp-hmm: Bayesian nonparametric hidden markov models with persistent states," Tech. Rep. 2777, MIT Laboratory for Information and Decision Systems, 2007.

[23] L. He, C. Zong, and C. Wang, "Driving intention recognition and behaviour prediction based on a double-layer hidden Markov model," Journal of Zhejiang University Science C, Issue 13, 2013, vol 3, pp. 208-217.

[24] S. Verwer, M. de Weerdt, and C. Witteveen, Learning "Driving Behavior by Timed Syntactic Pattern Recognition," In Proc. of the Twenty-Second international joint conference on Artificial Intelligence, 2011, vol. 2, pp. 1529-1534, doi: 10.5591/978-1-57735-516-8/IJCAI11-257.

[25] M. Kuhler and D. Karstens, "Improved Driving Cycle for Testing Automotive Exhaust Emissions," SAE Technical Paper Series 780650, 1978, doi:10.4271/780650.

[26] X. Huang, Y. Tan, and X. He, "An Intelligent Multifeature Statistical Approach for the Discrimination of Driving Conditions of a Hybrid Electric Vehicle," IEEE Transactions on Intelligent Transportation Systems, Dec. 2010, pp. 1-13.

[27] A. M. Liu, "Modeling Differences in Behavior Within and Between Drivers, Human Modelling in Assisted Transportation (Models, Tools and Risk Methods)," 2011, pp. 15-22, doi: 10.1007/978-88-470-1821-1_3.

[28] J. C. van Gemert, C. G. M. Snoek, C. J. Veenman, A. W. M. Smeulders, and J. Geusebroek, "Comparing Compact Codebooks for Visual Categorization," Computer Vision and Image Understanding, 2010, vol. 14, iss. 4, pp. 450-462.

[29] S. Griesche, "Images in mind – Design metaphor and method to classify driver distraction in critical situations," DLR presentation in interactIVe project final event, http://www.interactive-ip.eu/publications, retrieved April, 2014.

[30] J. Lin and Y. Li, "Finding structurally different medical data," Proceedings of the Twenty-Second IEEE International Symposium on Computer-Based Medical Systems, August 3-4, 2009, pp.1-8, Albuquerque, New Mexico, USA.

[31] R. Rossetti, J. Almeida, Z. Kokkinogenis, and J. Goncalves, "Playing Transportation Seriously: Applications of Serious Games to Artificial Transportation Systems," IEEE Intelligent Systems, July-Aug., 2013, vol. 28, no. 4, pp. 107-112.

 ISBN: 978-1-61208-341-4

[32] TEAM IP, Deliverable D1.0, “TEAM users, stakeholders and use cases,” www.collaborative-team.eu, retrieved: April, 2014.

[33] TeleFOT project, Seventh FP, co-funded by the European Commission DG Information Society and Media within the strategic objective “ICT for Cooperative Systems”, 2008-2012, http://www.telefot.eu/pages/index/?id=43, retrieved May 2014.

[34] Field experiment on intelligent speed adaptation (Tele-ISA), VTT research project, 2009, http://www.lintu.info/hanke_TeleISA.htm, retrieved: May 2014.

[35] Trafisafe, ITS safety project, http://www.trafi.fi/turvallisuus/trafisafe, retrieved May 2014.

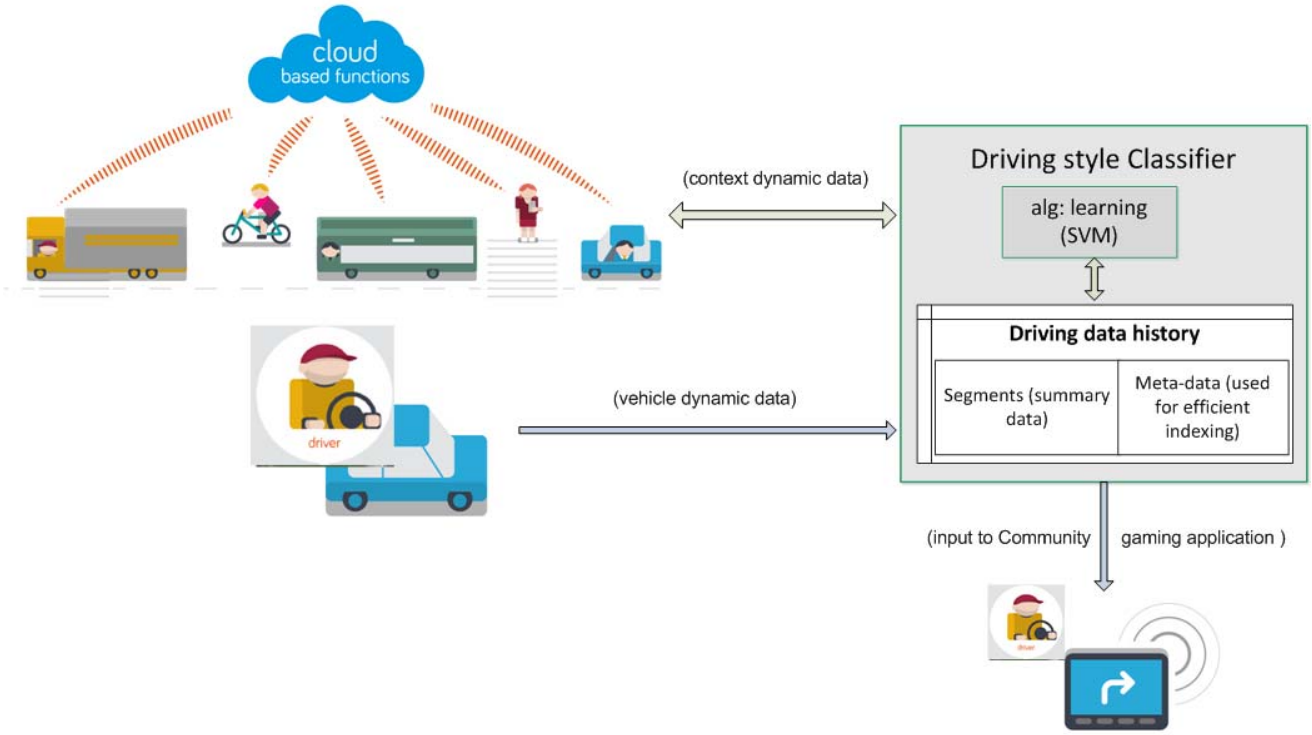


Figure 1: Functional concept diagram for a collaborative driving serious gaming application based on driving style classification

 ISBN: 978-1-61208-341-4

A Dynamic Service Module Oriented Framework for Real-World Situation Representation

Peter Halbmayer and Gerold Hoelzl and Alois Ferscha
Institute for Pervasive Computing
Johannes Kepler University Linz
Linz, Austria
{halbmayer, hoelzl, ferscha}@pervasive.jku.at

***Abstract*—The maintenance of context information for real-world environments contains several challenges when it has to be computationally observed. Any event that is observable in the real-world has to be registered and may lead to transitions in the digital system. Therefore, a representation of the environment, the affected users, their whereabouts and their interactions with the system is required. A software framework is presented that provides a generic set of methods to collect information from multiple, heterogeneous sensors deployed within the environment. A non-deterministic communication topology is established, which handles a distributed version of a system state on a best effort basis. The system is designed to be potentially applied within various environments and the primary application scenario is represented by the implicit energy management in single-family homes. There the practical deployment of the system proves the usability and sustainability of the presented approach.**

***Keywords*—Dynamic Device and Service Discovery; Dynamic Context Recognition; Adaptive System Behavior; Flexible Power Management; Opportunistic Sensing.**

I. INTRODUCTION

The PowerIT system [1][2] is a development that provides implicit energy management in households on the behalf of users living in the environment. For this purpose, different aspects need to be handled in real-time like the activities users perform, the situations they are in, the energy that is drawn by household gadgets and also the interaction with notification and services is an important aspect.

The main contribution of the system is the processing of dense context information for practical and ubiquitous digital system operation without explicit user interaction. An important aspect in this sense is to provide unobtrusive operation in the background for optimal and convenient system utilization. Thus the digital system moves to the background and the users focus is directed to actual every day tasks at hand instead of being concerned with system maintenance.

The deployment diagram depicted in Figure 1 presents a schematic overview of how the system is generally setup. A base infrastructure that is comprised of a Home Server (HS) that is connected to a set of energy meter and control devices, build the core of the system. The energy meter and control devices are implemented as wall plug outlets that are used to connect any household gadgets, measure their energy consumption and switch their state on or off. The user is integrated into the system by collecting and evaluating data from the mobile devices that are carried along. A typical system installation consists of 20 wall plug outlets, a single HS, and a smart watch - smartphone/tablet combination for every participant which lies around four.

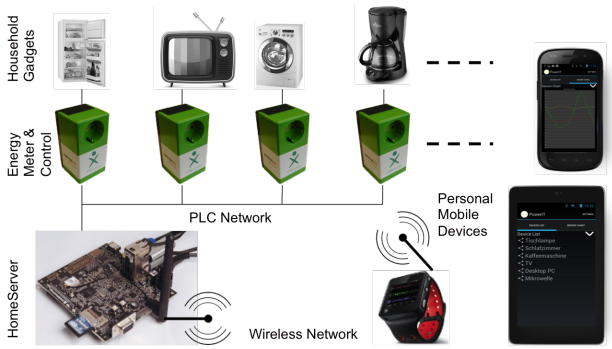


Figure 1. Deployment Diagram

The connection of mobile devices (e.g., smartphone or smart watch) to the base infrastructure is of a dynamic nature as users enter and leave the environment in an unpredictable way. This is also true for interconnections between mobile devices themselves (e.g., user take smartphone away and grabs it later). Therefore, such cases have to be handled with special precautions. The corresponding underlying network has to adapt itself automatically for seamless service provision.

From this description, the main aspects the system has to handle in software can be drawn. They are:

(i) Activity and context recognition
(ii) Opportunistic sensor and actuator management
(iii) A rule engine for implicit control.

Dynamic and adaptive sensor and actuator management builds the base for context recognition and implicit control. When an event is sensed (e.g., a user leaving the house), control mechanisms have to trigger to switch the state of devices (e.g., turn off all lights). To achieve the transport of corresponding messages and commands, the communication topology must be available to guarantee the successful delivery of messages. The dynamics of users have to be maintained in the system by unregistering users when leaving and (re-) registering them when they return.

As the practical usability of the system was one of the main interests in our project, it was necessary to determine the most important aspects to obtain an implementation that can withstand real-world conditions. In a previous project [3], where similar studies were conducted on a smaller scale than in PowerIT, it has been seen that certain characteristics need

 ISBN: 978-1-61208-341-4

to be regarded to ensure the success of such a system. This was learnt from user questionnaires as well as practical system application in real-world sites.

Qualitative requirements that are necessary in such a system are

(i) The recoverability and fault tolerance of the system,
(ii) The unobtrusive operation and minimization of configuration and maintenance steps and
(iii) The organized reporting of failure in case of unrecoverable states entered.

A software model that addresses these aspects is required for practical realization of the system. Functionalities to build up basic communication streams, analyze collected data in real-time and forward control commands to dedicated endpoints are required. Therefore, in the following sections, first, related work is reviewed which is followed by a detailed system architecture description comprised of a system overview, system dynamics regarding the roles of devices and temporary registration, as well as activity recognition for system control. The final conclusion evaluates the results found in this work.

II. Related Work

The requirements mentioned in the last section are of vital importance for the realized system. Existing work already covers the core technology that is necessary to let devices exchange information between each other on an ad-hoc basis. Here, part of this technology is used in a practical context and concerned with the application of the implemented methods in real-world deployments. Building up on this base, a dynamic middleware was developed that (i) is platform independent and (ii) autonomously reconfigures itself dependent on actual states of subsystems and events in the current environment.

The base infrastructure in a household provides the backbone regarding the energy management concerned for the particular installation site. System dynamics are mainly represented by inhabitants appearing and disappearing throughout the day and the sensor traces they leave. These events are registered, and system components will get adapted accordingly (e.g., switching off unneeded devices currently in standby). It is common nowadays that a user is equipped with more than one personal device (e.g., smartphone and smart watch) where device interactions have to be defined. The influence of the developed system goes further in the way that part of these dynamics will also get reflected on the mobile devices the user carries around whether at home, at work or in times when any leisure activities are performed.

Therefore, the essential aspects the system is concerned with, are multi platform capability (to enable widespread deployment), different interconnection constellations (given by dynamic system behaviour over time), as well as general adaptation over time by recognizing and mapping behaviour and preferences of users.

A. Dynamic Module Systems and Service Platforms

At the core of the system it was required to enable system execution on different end devices. This ranges from conventional desktop systems to mobile smartphone and tablet devices. A modular component setup was specified for which the OSGi framework reference implementation Felix [4] was used. Modular system specifications have been addressed in various literature where multi-layered, service oriented software has been developed for application in different domains like telematics [5], web technology [6], cloud services [7], health and elderly care [8], vehicular network management [9] or context computing [10], in general.

B. Device and Service Discovery

Device and service discovery is of major interest in all cases where distributed entities have to exchange information on behalf of an unreliable and dynamic connection infrastructure. In the PowerIT system, the dynamic part mainly consists of the interactions of the mobile devices (e.g., smartphone and smart watches of a user) amongst each other, and with the infrastructure. Also, the change of a user between multiple infrastructures (e.g., from home to work) has been considered as an extension.

Depending on the connectivity to the system, the state of users and present and upcoming interactions, the system needs to adapt to these conditions. Although the main infrastructure stations might be known in advance, for mobile entities, this is not the case throughout the whole lifetime of the system. Therefore, device and service discovery has been utilized much in the manner of [11][12] where standard internet protocol (IP) services are applied for this purpose.

C. Opportunistic sensor configuration, activity recognition and decision making

For automatic control, the activities of users get observed and are distributed accordingly in the system to enable the switching of devices or groups of gadgets. For this purpose results found in [13][14][15] can be utilized by establishing recognition chains for every sensor data input stream that is of interest and importance for the system functionality. This issue is addressed in more detail in the architecture technology section. After activities, or a change of activities has been observed from sensor data, this information is forwarded to the system where it gets decided if it will eventually lead to any state change or not. This is achieved by putting the results of Kurz et al. [16] into practical application. Within our first test setups it has been shown that this approach already shows positive impact although it was deployed only for a controlled part of the system to prevent a degradation of overall system usability. The decision module that is responsible to determine if any input data will lead to an actual actuator control command is reused from the work in [17]. A generic set of rules that is capable of integrating arbitrary sensor inputs dynamically to build up statements about the system state was realized and successfully brought to practical application.

III. System Architecture

The intention for the system was to have an unobtrusive setup of personal information and communication technologies (ICT) services that allow the recognition of user centered activities of daily living to control the electrical system in its surroundings. It has soon been detected by the authors that for a system that has to provide its functionality in a 24/7 fashion, special precautions needed to be taken to serve the

 ISBN: 978-1-61208-341-4

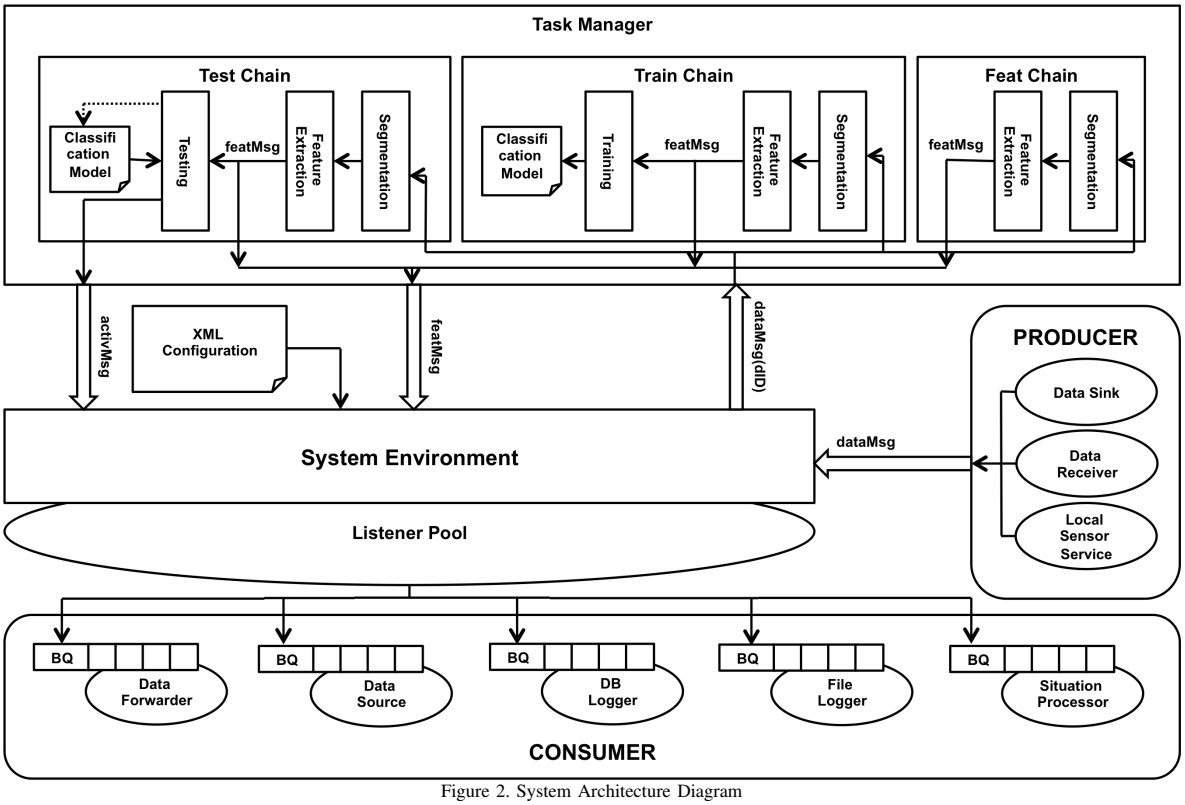


Figure 2. System Architecture Diagram

given requirements. The system needed to be implemented in a way, so that failures on a long term basis (e.g., unforeseen bugs in rarely called functions or accumulated and never freed memory) will be recovered automatically, unrecognized by the user in the optimal case.

From this description it is clear that the system depends on several parameters that can be divided into options that are static throughout the lifetime of the system, and others that change over time dependent on various parameters like the time of day, location and activity of a user, but also the intentions of a user with respect to electric device usage, convenience functions and real-world situation context (e.g., times when system automatisms are unwanted).

The specified system architecture results in a generic framework that can be executed on various devices (like smart watches, smartphones, tablet computers or desktop systems) running different operating systems. It is implemented in the Java programming language, and is, therefore available for all environments capable to execute a Java Runtime Environment (JRE) [18]. By relying on the OSGi middleware framework, a better separation and modularization of system components is reached. This supports the execution of the framework under different role settings that are addressed below in Section III-B.

A. System Overview

```
<config>
(a)     <system>
                <devicename>MotoACTV</devicename>
                <is-log>true</is-log>
                <is-act-rec>true</is-act-rec>
                <local-sensor>true</local-sensor>
        </system>
(b)     <fwds>
                <fwd>
                        <name>GGPH</name>
                        <type>feat</type>
                        <conn>bt</conn>
                        <mac>AC:22:0B:A4:22:93</mac>
                        <uuid>00001101-0000-1000-8000-00805F916001</uuid>
                </fwd>
        </fwds>
(c)     <rcvs>
                <rcv>
                        <name>HS</name>
                        <type>feat</type>
                        <conn>ip</conn>
                        <ip>10.0.0.1</ip>
                        <port>16001</port>
                </rcv>
        </rcvs>
(d)     <snks>
                <snk>
                        <name>GGPH-ctrl</name>
                        <type>ctrl</type>
                        <conn>bt</conn>
                        <uuid>00001101-0000-1000-8000-00805F918001</uuid>
                </snk>
        </snks>
(e)     <srcs>
                <src>
                        <name>GGPH-cnt</name>
                        <type>cnt</type>
                        <conn>ip</conn>
                        <port>22001</port>
                </src>
        </srcs>
(f)     <loggers>
                <logger>
                        <name>wws</name>
                        <type>file</type>
                </logger>
                <logger>
                        <name>feat</name>
                        <type>file</type>
                </logger>
        </loggers>
</config>
```

Figure 3. Main configuration sections

The system architecture is divided into three main categories that are enlisted below. A graphical representation of the system architecture is shown in Figure 2. There, the system components available locally on a host node are depicted.

1) The base functions like configuration handling, message passing and thread pools.
2) Communications including device and service discovery and transmission of different message types.
3) Activity recognition, locally and remotely.

The system core consists of a structure for managing the runtime environment of the framework which also includes a listener pool to which elements interested in any specific messages register. To forward and distribute messages within the system a producer - consumer pattern is specified where

producers offer data to the system from local or remote sensor sources and consumers forward data to remote hosts or are responsible for local persistence. Activity recognition is defined in terms of a task chain that contains data segmentation, feature extraction, training and testing of classification models. Multiple task chains can be handled concurrently on a single host.

Depending on the configuration of every host node, an end device can take a different role, which is explained in the following subsection. A simple example configuration is presented in the listing in Figure 3.

Initially, a communication system that allows the propagation of sensor data from low power embedded devices up to full-blown server installations to access the data at different device instances with varying computation performance was designed and developed. The system implements general patterns that allow the execution of the same software on different host nodes with varying configurations.

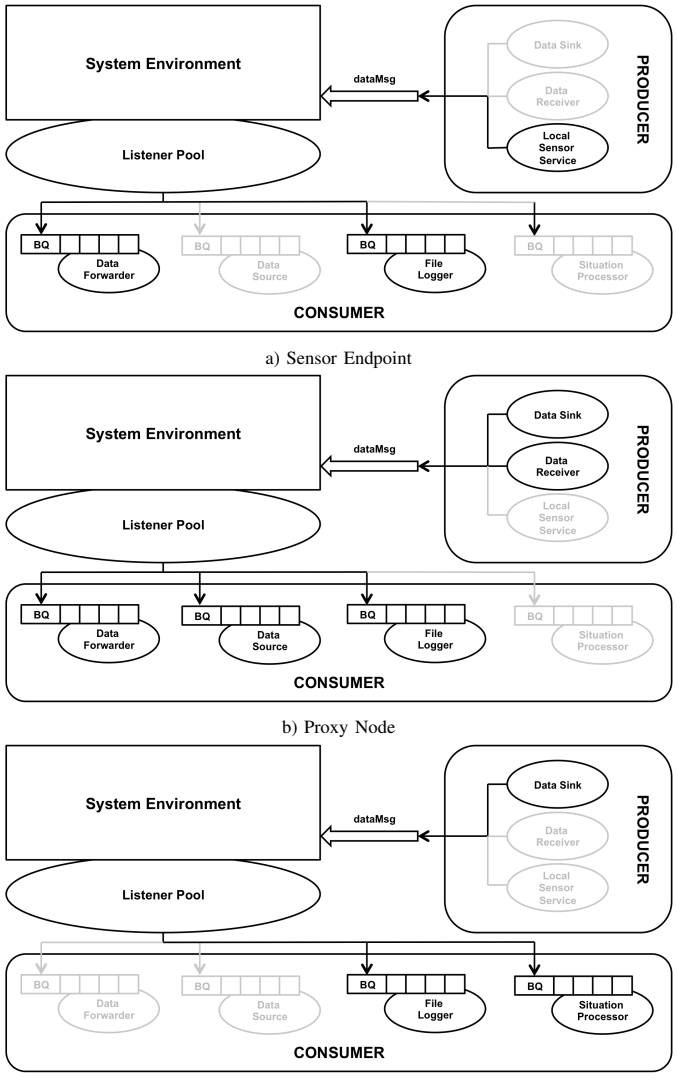
Starting with an extendable configuration, devices have the capability to re-configure themselves dependent on other devices found in the environment and the role the local device has to full fill dependent on the global system state.

The configuration depicted in Figure 3 represents a default configuration that illustrates possible settings. It is used as example to enlist the main configuration items that will get deployed across the nodes included in the system. The first section of the listing, named (a) *system*, defines general system properties which inform if local sensors (if available), file logging and activity recognition are enabled. This already partially defines the role (cf. Section III-B) of the host, on which this configuration item is deployed. Without any connection configurations only local operation would be possible but for full adaptive participation in the system the corresponding communication channels need to be stated. The relevant items for this purpose in the configuration file are found under the sections (b) *fwds*, (c) *revs*, (d) *snks* and (e) *srcs*, respectively. Although it is not shown in the listing, it is possible to setup several connections of every type together to form the different roles required in the system. Additionally, as depicted in Figure 3, it is possible to drive different connection types simultaneously which even extends the number of possibilities how system parts can be connected together.

Section (f) *loggers* enables the corresponding logging capabilities where the subsections determine which types of messages are logged. In the above example, raw data (for later offline analysis) as well as feature data derived from raw data are logged. It is possible to log other data types like preprocessed raw data or activities retrieved from the recognition functionality.

By using this definition for system entities, a message passing system in the style of a Model-View-Controller (MVC) [19] design pattern has been specified. To evaluate this design, but also to serve as a workhorse for data acquisition, the implemented system has been deployed and put in operation in three installation sites. From the configuration items presented in the listing, it can be seen that IP, as well as bluetooth based connections are possible (as depicted in configuration sections (b) and (c)).

B. Dynamic Device Roles



a) Sensor Endpoint

b) Proxy Node

c) Home Server

Figure 4. Dynamic Device Roles

The system provides multiple sub services on different end devices. With this approach, services can be offered that optimizes system utilization regarding the resources of the runtime device. This way it is possible to keep the data source on the smart watch device while interaction and control can be handled over to a remote device like a smartphone. An accompanying advantage of this design is that the same software packages can be reused on different end devices to provide different services corresponding to the used configuration.

Device roles are tightly dependent on the functionality that an entity in the system will perform and the type of connection required for data exchange. Communication is set up in a server - client fashion and dual roles for servers as well as clients exist to form proxy devices. One type is the *sink* server to which a corresponding *forward* client pushes data. The other server type is called *source* that that provides data to be fetched by *receiver* clients. By this approach, any device in the system can flexibly be configured to play any role, which is autonomously reconfigured during runtime if

 ISBN: 978-1-61208-341-4

the purposes of a device changes (i.e., switching from an end device configuration to a proxy configuration as another end device requests this feature temporarily).

The general idea is that temporal unreachability or failure of one device will get detected by the system and another device in the system formation will be adapted to take over the role of the failed device. This was one reason for the unified software implementation as it can be guaranteed that the replacement device is able to handle the same functionality. The roles that are relevant in our setting are:

1) Sensor endpoint
2) Proxy
3) Local (installation site) server, Home Server
4) Global server

Corresponding schematics are depicted in Figure 4. In our system, sensor endpoints (Figure 4a)) are the energy meter devices and smart watches. Smartphones are utilized as proxy devices (Figure 4b)) and the central access point is represented by the Home Server (Figure 4c)). The functions that are necessary on the corresponding type of device are depicted in bold in the respective subfigure, the illustration concerned with activity recognition is omitted there.

C. Activity Recognition

The dynamic nature of system connections turned on the requirement that the system component performing activity recognition and tracking has to be capable of these dynamics as it was necessary to rapidly switch devices that are executing recognition chains. For example, when a smartphone performs activity recognition for the local sensor, as well as for data from a remote smart watch sensor and the user leaves the smartphone back while keeping the watch on, then, globally this task has to be split to both devices performing its own activity recognition each. If afterwards the user returns to the smartphone this needs to be detected by the system and recognition chains can be processed on the smartphone again.

Also, for practical reasons, the resources on a smart watch are much more limited than on a smartphone or on the Home Server especially regarding battery lifetime. As the activity recognition is a computationally expensive task, it might be convenient to outsource parts of this task from the device where the sensor data acquisition is performed. These cases are depicted in the upper third of Figure 2.

Activity recognition is defined in terms of task chains, where a single task chain is set up for every type of sensor and device. A task chain performs raw data segmentation, feature extraction, the classification of featured training data, and the evaluation of new data according to a set up classification model. Besides classification of raw sensor data, corresponding statistics are collected for every sensor of how long and how much a device is used for which purpose. This information is used to continuously update the corresponding background context to offer system services according to their utilization within the actual application scenario.

D. Registration, Un-Registration and Re-Registration

In a smartphone/smart watch or smartphone/smart watch/HS constellation, communication is always preceded with a discovery stage dependent on the actual states of devices. If, for example, active communication is ongoing, no discovery is necessary and can therefore be disabled at this time. If connections break suddenly, or the activity state of a user indicates a change within the network topology (e.g., because of change in location), a re-scan of the environment will become necessary. Therefore, occasion based device and service discovery is implemented to execute the process only when the system is in a certain state and disabled in all other cases. The main intention for this approach is to save battery lifetime of mobile devices.

When device and service discovery is processed, potential remote nodes get detected and their supported services registered. On initial system startup a device and service discovery is performed where found nodes and their capabilities are put into a local cache. At subsequent connection attempts, this cache is first iterated over to reestablish well-known connections. Only if this is not possible new device and service scans are started. This process is depicted in the diagram in Figure 5. For this purpose, JMDns [20] is utilized as it has proven to be a simple, efficient and reliable solution that is able to provide this functionality at a sufficient degree.

After the temporal topology is determined the self descriptions of newly found nodes are exchanged. The configuration exchange function is a service every device supports to be able to participate in system communication. The configuration items can also be cached to avoid the explicit exchange for this item. After remote configurations are known, any required reconfiguration is handled.

To determine which device is responsible for which task, preferences are defined. If two devices share the same preference for a certain function, a default preference is given by the device roles in the order of their enlisting. Depending on the specific task the preference for a role is higher as for another. Practically, it is better to perform calculations locally and only transmit results than forwarding raw data in a stream like fashion which has severe implications on the battery life times in case of mobile devices.

IV. Conclusion

In this work, it has been shown that the existing problem of integrating heterogeneous technical entities in digital environments can be replied by (i) the establishement of conventions regarding the definition of concerned items, (ii) specification of communications, and (iii) information representation within its actual context. A generic architecture approach has been presented that allows the execution on a variety of platforms and maps platform features (i.e., present sensors, actuators or processing units) onto corresponding input and output channels. This way, device interaction can be achieved to perform information exchange for a variety of purposes in an adaptive manner.

The necessity of scaling up the number of items and involved people will appear when the system is going to be installed in broader context environments like workplace (e.g., office or factory) or leisure sites. These cases are considered in the framework implementation and need deeper research under practical real-life conditions.

 ISBN: 978-1-61208-341-4

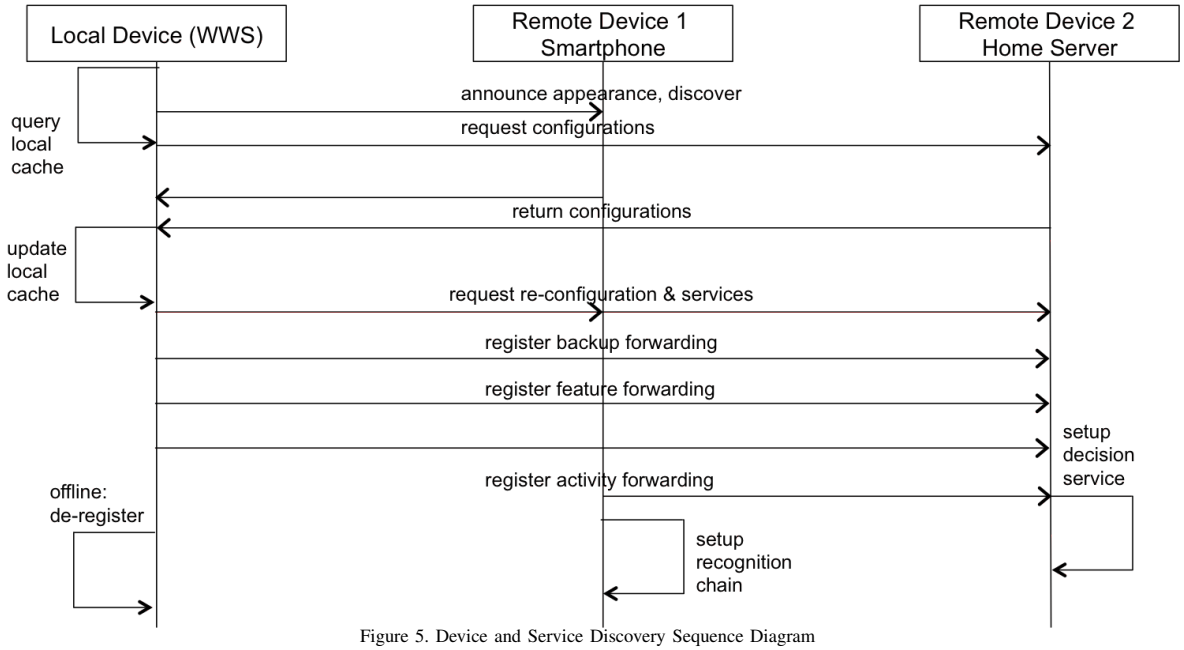


Figure 5. Device and Service Discovery Sequence Diagram

The requirements identified in the introduction section were shown to be solved by the implemented methods. The developed approach is empirically tested to gain real-world evidence within the PowerIT project at the moment. Utilizing the test installations, it turned out that the implemented services and modules are performing as expected. The real world installations are an ongoing and evolving process yielding to new findings for future enhancements of the developed framework. Consequently, it was shown that the application of our dynamic system for context and situation representation heads in the right direction.

ACKNOWLEDGMENT

The project PowerIT acknowledges the financial support of the FFG FIT-IT under grant number: 830.605.

REFERENCES

[1] G. Hoelzl, P. Halbmayer, H. Rogner, C. Xue, and A. Ferscha, “On the utilization of smart gadgets for energy aware sensitive behavior,” in The 8th International Conference on Digital Society, ICDS 2014, March 23 - 27, Barcelona, Spain, March 2014, pp. 192–198.

[2] G. Hoelzl et al., “Locomotion@location: When the rubber hits the road,” in The 9th International Conference on Autonomic Computing (ICAC2012), San Jose, California, USA, September 2012, pp. 73–78.

[3] A. Ferscha, J. Erhart, P. Halbmayer, M. Matscheko, and M. Wirthig, “Powersaver - activity-based implicit energy management,” in 15th International Symposium on Wearable Computers (ISWC2011), June 2011.

[4] “Apache Felix,” http://felix.apache.org/, [retrieved: 05, 2014].

[5] Y.-L. Chu et al., “An integrated java platform for telematic services,” in Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on, 2010, pp. 590–593.

[6] D. Carlson, B. Altakrouri, and A. Schrader, “Ambientweb: Bridging the web’s cyber-physical gap,” in Internet of Things (IOT), 2012 3rd International Conference on the, 2012, pp. 1–8.

[7] F. Houacine, S. Bouzefrane, L. Li, and D. Huang, “Mcc-osgi: An osgi-based mobile cloud service model,” in Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on, 2013, pp. 1–8.

[8] K. C. Kang, S. U. Heo, and C. S. Bae, “Android/osgi-based mobile healthcare platform,” in Advanced Information Management and Service (ICIPM), 2011 7th International Conference on, 2011, pp. 125–126.

[9] T.-W. Chang, “Android/osgi-based vehicular network management system,” in Advanced Communication Technology (ICACT), 2010 The 12th International Conference on, vol. 2, 2010, pp. 1644–1649.

[10] D. Carlson and A. Schrader, “Dynamix: An open plug-and-play context framework for android,” in Internet of Things (IOT), 2012 3rd International Conference on the, 2012, pp. 151–158.

[11] R. Klauck and M. Kirsche, “Bonjour contiki: A case study of a dns-based discovery service for the internet of things,” in Ad-hoc, Mobile, and Wireless Networks, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7363, pp. 316–329.

[12] R. N. Lass, J. Macker, D. Millar, and I. J. Taylor, “Gump: Adapting client/server messaging protocols into peer-to-peer serverless environments,” in Proceedings of the 2Nd Workshop on Bio-inspired Algorithms for Distributed Systems, ser. BADS ’10. New York, NY, USA: ACM, 2010, pp. 39–46.

[13] D. Roggen, K. Förster, A. Calatroni, and G. Tröster, “The adarc pattern analysis architecture for adaptive human activity recognition systems,” Journal of Ambient Intelligence and Humanized Computing, 2011, pp. 1–18.

[14] G. Hoelzl, M. Kurz, and A. Ferscha, “Goal processing and semantic matchmaking in opportunistic activity and context recognition systems,” in The 9th International Conference on Autonomic and Autonomous Systems (ICAS2013), March 2013, pp. 33–39.

[15] ——, “Goal oriented recognition of composed activities for reliable and adaptable intelligence systems,” Journal of Ambient Intelligence and Humanized Computing (JAIHC), July 2013, p. in Press.

[16] M. Kurz, G. Hoelzl, and A. Ferscha, “On the utilization of heterogeneous sensors and system adaptability for opportunistic activity and context recognition,” in Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2013), May 27 - June 1, 2013, Valencia, Spain, May 2013, pp. 1–7.

[17] “JRuleEngine - OpenSource Java Rule Engine,” http://jruleengine.sourceforge.net/, [retrieved: 05, 2014].

[18] “Java Standard Edition,” http://java.oracle.com/, [retrieved: 05, 2014].

[19] G. E. Krasner and S. T. Pope, “A cookbook for using the model-view controller user interface paradigm in smalltalk-80,” J. Object Oriented Program., vol. 1, no. 3, Aug. 1988, pp. 26–49.

[20] “Java Multicast DNS,” http://jmdns.sourceforge.net/, [retrieved: 05, 2014].

 ISBN: 978-1-61208-341-4

Performance Evaluation of Reconfiguration Algorithms for the Reconfigurable Network on Chip Architecture RecMIN

Alexander Logvinenko, Dietmar Tutsch
University of Wuppertal
Emails: alexanderlogv@gmail.com, tutsch@uni-wuppertal.de

***Abstract*—The Reconfigurable Multi-Interconnection Network (RecMIN) is a new network architecture that reduces inefficiency and increases the throughput of the network on chip. The RecMIN topology adapts itself to traffic flow by reconfiguration. Three reconfiguration algorithms are employed, in order to take advantage of the capabilities of the RecMIN architecture. The η-algorithm, the minimal queues algorithm and the pattern identification algorithm allow the network to adapt itself to different traffic distributions. Furthermore, an observation technique that notes changes in traffic pattern is presented, in order to avoid infinite reconfiguration processes. The performance of the algorithms is presented.**

Keywords-Network on Chip; Reconfiguration Algorithms; Reconfiguration Architecture.

I. INTRODUCTION

Modern Systems on Chip (SoC) are built so that they consist of many independent individually designed units (Intellectual Property cores or IP- cores), e.g., cache memory, I/O controllers, audio/video interfaces, etc. Buses were used up to now in order to enable communication among these units. Today, however, designers prefer Networks on Chip (NoC) for efficient interaction among IP-cores. Therefore, the speed and efficiency of modern SoC depend not just on the speed of single units of IP but also on the properties of the NoC used [1]. The main properties of the NoC are source output, target throughput and packet delay. The latter ones depend not just on topology of the network, routing algorithm, buffering strategy, packet switching but also on how efficiently network operates in case of bottlenecks during the packet traffic flow.

The popular solution to solve the problem of a partially overloaded network (bottlenecks) due to inefficiency, is to implement a complex algorithm that reroutes data flow in NoC. The complexity of such algorithms usually grows exponentially with the size of network. So, as an alternative to the rerouting algorithms, some works from the academic community have been focusing on the possibility of adopting NoC by reconfiguration.

For instance, Tutsch and Lüdtke [2][3][4] and Al Faruque [5][6] suggest that the directions of data flow should be changed in order to optimize the NoC for special traffic profiles. Unlike them, this paper continues the previously [7][8][9] introduced topic of the RecMIN architecture. In this article, however, we present three different algorithms for the optimization of a network that uses the RecMIN architecture.

The paper is structured as following: Section 2 introduces the reconfiguration architecture RecMIN. Section 3 deals with the η-function, which enables to evaluate of the network-on-chip performance. In most important Section 4, three algorithms are presented and compared: η algorithm, minimal ques algorithm, pattern identification algorithm. Section 5 concludes.

II. RECONFIGURATION ARCHITECTURE RECMIN

The main problem of NoC as compared to the full connection of all inputs/outputs is the chance of bottlenecks to arise given certain traffic structures. In this work, Multistage Interconnection Metwork (MIN) architecture is used, which is built out of 2×2 routers [10]. An example of this kind of network is shown in Fig. 1. The technical realisation of MIN topology is given, e.g., in [11] and [12]. One of the characteristics of MIN is that all the traffic loads have to pass through all the stages of the MIN. Especially for asymmetrical traffic, the connection wires between the stages can lead to tailbacks.

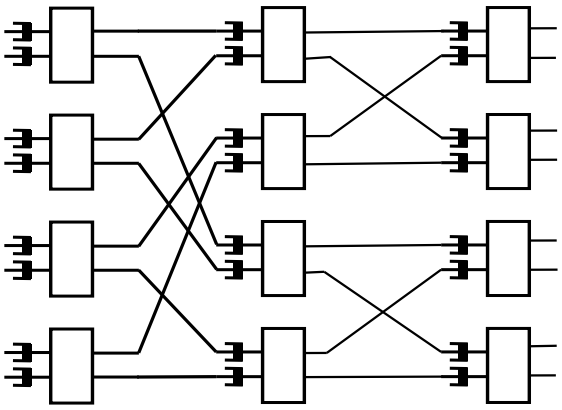


Figure 1. MIN architecture with 8x8 inputs/outputs built out of 2x2 routers

Reconfiguration architecture RecMIN solves the problem of bottlenecks in two out of three possible cases. The proposal is to create the MIN not from the 2x2 routers, as usual, but from specific reconfiguration half cells - Reconfiguration Half Cell (RecHC). The architecture of this cell is given in Fig. 2.

RecHC has 8 inputs and 8 outputs. In front of each input, one buffer element is located. Each half-cell can be used in one of these two possible modes: In the first mode (Mode A), the RecHC consists of four independent 2x2 routers. In the

 ISBN: 978-1-61208-341-4

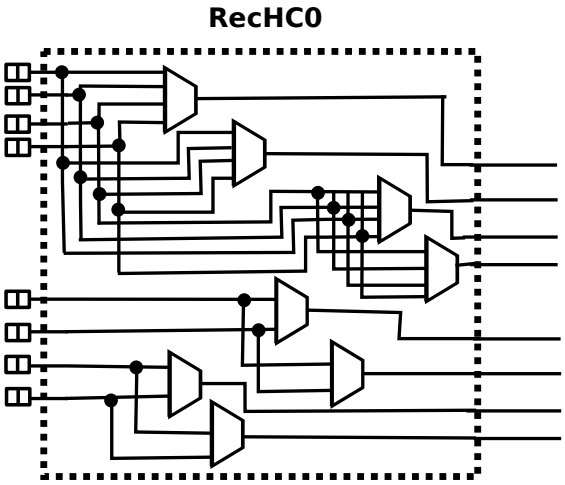


Figure 2. The architecture of reconfiguration half cell - RecHC

second mode (Mode B), there is however just one 4x4 router in the upper part of the cell, and four simple wire connections without any logic in its bottom part. If a RecHC changes the mode from A to B (Fig. 3), then packets which arrive in the upper part of the Half Cell are distributed correctly without problems. Though, in the bottom part of the RecHC problems may arise, since in the mode B no redirection takes place, and the packets are transferred straight forward (Fig. 3). For example, after switching from mode A to mode B some packets in buffer of input i4 that are addressed to output o5 have no possibility to arrive at their targets (e.g., IP-cores). Therefore, usage of two half cells simultaneously is to be preferred.

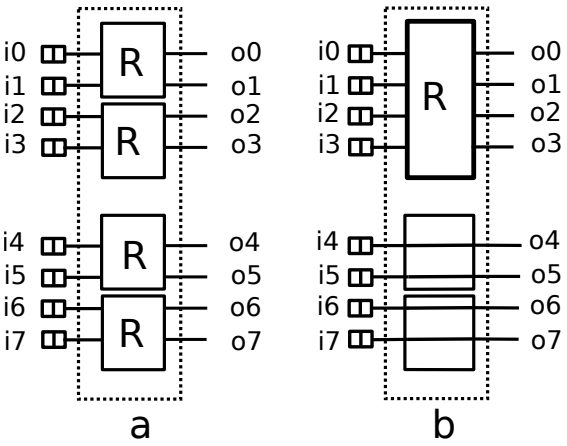


Figure 3. RecHC in two modes. a: mode A, b: mode B

The two RecHCs are put together (the second one upside down), to form one reconfiguration cell - RecCell (Fig. 4). If both of RecHCs that build RecCell are put into the Mode A, then the construction leads to two independent MINs (Fig. 4a), with 4x4 inputs-outputs and 2x2 routers each. If the two RecHCs are put in mode B, then two independent 4x4 routers emerge (Fig. 4b). The other two combinations (AB and BA) are meaningless and therefore are not used. So, a full cell has two possible reconfigurations: folded (BB) and unfolded (AA).

With RecCells, it is possible to build a MIN. The resulting structure is called RecMIN. If the number of 2x2 switches

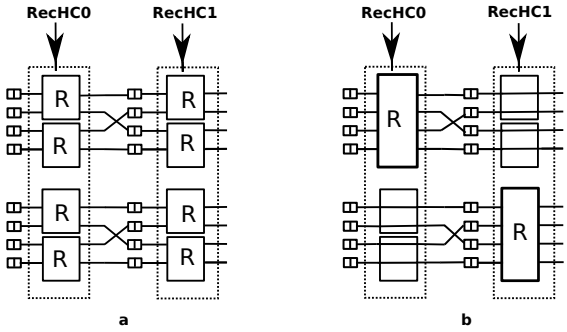


Figure 4. RecCell in two modes. a: unfolded mode, b: folded mode

in MIN is divisible by 16, the entire network can be built from reconfiguration cells. Otherwise, it is necessary to use two non-reconfigurable 2x2 switches in order to connect the reconfiguration cells. Therefore, RecMIN with an arbitary number of 2x2 routers can be implemented.

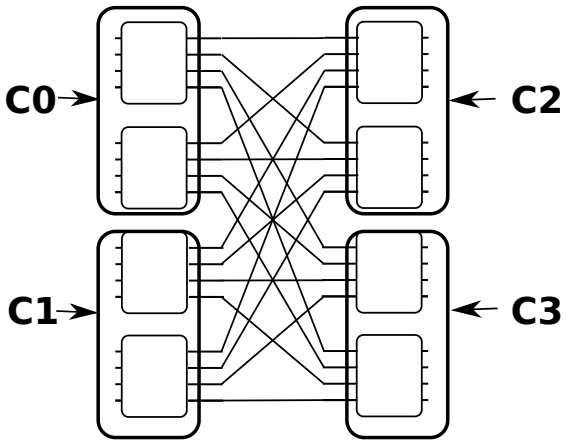


Figure 5. RecMIN with 16 inputs/outputs

In this paper, the RecMIN with 16 inputs outputs is used as an example for RecMIN architecture (Fig. 5). This RecMIN can be build out of four RecCells: C0, C1, C2 and C3.

If we compare our architecture (Fig. 6) with the one of the non-reconfigurable MIN with 2x2 routers, we will see that the dotted line marked router connection (between the first and the second stage of the 2x2 routers) can be reconfigured. So, if the traffic in the network generates bottlenecks in these places, the NoC can reconfigure its topology according to the adaptation of the architecture to the traffic load, and so increase the throughput of the network and decrease the packet delay.

It can be said that if the traffic unfortunately generates a bottleneck in one of the non-reconfigurable wires, the reconfiguration will not help. But, usually, the designer of the NoC knows the application for which the network is to be designed, and so can pre-arrange the most expected bottleneck-wires inside the reconfiguration cells.

The other way around, the 4x4 router would have less throughput than a 2x2 router [13]. So, for the symmetrical high load (more than 0.63 flits per clock cycle), the 4x4 routers will automatically become NoC bottlenecks. In this case, a back

 ISBN: 978-1-61208-341-4

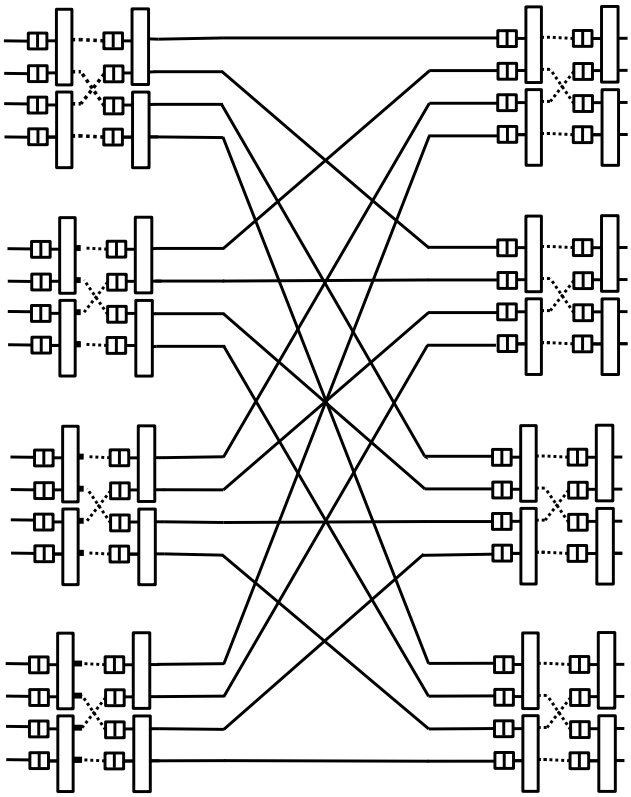


Figure 6. RecMIN with 16 inputs/outputs

reconfiguration of the RecCells to unfolded mode is necessary (see [13]).

The other disadvantage of the architecture is that two independent routers of the NoC are now bonded together. If one of the 4x4 routers of RecCell "decides" to change mode (from unfolded to folded or other way around), it has to check if the other router of the same RecCell will "agree" to change the mode as well.

III. Evaluation of Network Performance

The network performance of asymmetrical NoC for asymmetrical load is measured by three main parameters: throughput of network sources (ς_i, where i is a number of the source in NoC), throughput of network targets (τ_i, where i is a number of the target in NoC), and packet delay for each target (δ_i, where i is a number of the target in NoC). To evaluate network efficiency dependent on the packet load, we use η-function:

$$\eta = \sum_{i=0}^{N-1} (\varsigma_i * C_{\varsigma_i} + \tau_i * C_{\tau_i} + \delta_i * C_{\delta_i}) \qquad (1)$$

where N is the number of sources/targets in NoC and constants C_{ς_i}, C_{τ_i}, C_{δ_i} are priority weights for throughput and delay defined by SoC designer. By setting the priorities the designer specifies how important the corresponding NoC parameter is.

For example, for specific NoC the throughput for sources and targets may be not as important as a minimal delay. Furthermore, packet delay is especially important for the targets T_4 and T_5. In this case, the constants C_{ς_i}, C_{τ_i}, C_{δ_i} can be adjusted as follows:

$$\begin{aligned} C_{\varsigma_i} = C_{\tau_i} = 0 \quad & [\text{clock cycles/flit}] \quad \text{for } i \in \{0,..,N-1\} \\ C_{\delta_i} = -1 \quad & [\text{clock cycles}]^{-1} \quad \text{for } i \in \{0,..,N-1\} n \{4,5\} \\ C_{\delta_4} = C_{\delta_5} = -2 \quad & [\text{clock cycles}]^{-1} \end{aligned} \qquad . \qquad (2)$$

The parameter ς_i, τ_i, δ_i depend on the topology of the network, and are calculated using simulation. By simulating the different network reconfigurations, designer is able to compare the performance of different network topologies for specified traffic. Table 1 gives an example for a network with 16 inputs and 16 outputs consisting of four RecCells (Fig. 6). Constants of priorities are chosen according to (2).

In Table 1, G_i is the notation for each source (generator) i; P_{tr} (G_i) is the probability that the source i sends a packet per clock time unit; P_{rec} (T_x) is the probability that the target node x receives a packet from the generator i (G_i) per clock time unit.

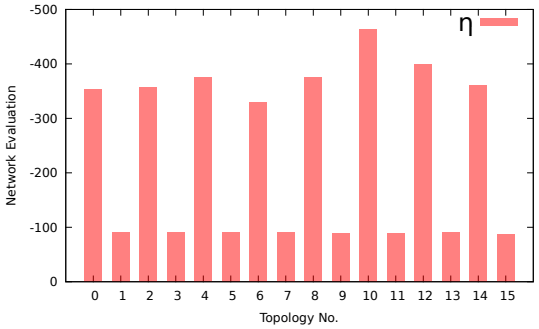


Figure 7. η-function for 16x16 RecMIN loaded with traffic from Table 1

The simulation results are presented in Fig. 7 (Simulation parameters for all simulations presented in this paper are: buffer size 16 phits for each buffer, conflict resolution algorithm for each router is "random choice", each packet consists of one flit, a flit equals the size of a phit). It shows the evaluation of different RecMIN reconfigurations, resulting from all possible RecCell modes. The used NoC consists of 4 RecCells each of them can be used in two possible modes, so, for this kind of network there exist $2^4 = 16$ possible topologies. As is shown in Fig. 7, η-function has the highest rates for topologies with an odd number (1,3,5 etc.), and the lowest rate for topology 10. Thus, for the optimal communication of NoC components by traffic defined in Table 1, RecMIN must be reconfigured to topologies 1,3,5,7,9,11,13, or 15.

IV. Reconfiguration Algorithms

It is not sufficient only to offer a reconfigurable architecture when considering the reconfiguration of NoC as an opportunity to improve its efficiency and performance. A second step is required in order to take advantage of the capabilities of the architecture: employing algorithms that allow the network to

 ISBN: 978-1-61208-341-4

TABLE I. LOAD IN RECMIN

Generator	$P_{tr}(G_i)$	$P_{rec}(T_{10})$	$P_{rec}(T_{11})$	$P_{rec}(T_{12})$	$P_{rec}(T_{rst})$
G_0, G_1	0.4875	0.2/16	0.3	0.2/16	0.2/16
G_2, G_3	0,3875	0.2/16	0.2/16	0.2	0.2/16
G_4 - G_7	0,55	0.1	0.2/16	0.2/16	0.2/16
G_8 - G_{15}	0.2	0.2/16	0.2/16	0.2/16	0.2/16

adapt itself to different traffic distributions. In this paper, we propose several algorithms that were developed for RecMIN architecture: the η-algorithm, the minimal queues algorithm and the pattern identification algorithm.

A. General requirements for algorithms

The tasks of the algorithm responsible for the reconfiguration of the network can be divided into the following steps:

- Monitoring the trigger: tracking events or sequences of events, after which the algorithm has to decide about the reconfiguration of the network topology.
- Looking for bottlenecks: finding parts of the network that need to be changed due to reconfiguration
- Looking for alternative structure: finding a topology to substitute the previous one
- Processing the reconfiguration

Each of these steps should avoid high time consumption and should require simple calculation wherever possible. (Implementation of complex calculations in hardware, requires expensive chip area). Another key issue is the question of stability. It is necessary to avoid a situation where the algorithm constantly tries to optimize the network. Doing so the algorithm continually conducts endless reconfiguration processes, hence preventing the network from operating in normal mode. For example, such a problem can occur if reconfiguration algorithm is unable to find an unambiguously best network topology. Thus, after checking different network reconfigurations the found topology is still not optimal. This state directs to the retriggering of the algorithm thereby starting a new reconfiguration process. Therefore, no reconfiguration process should be started, if the algorithm is unable to find a better NoC topology for the traffic flow unless the network traffic changes. Consequently, it is essential not only to have a trigger to reconfigure the network, but also to implement an observation technique that notes changes in traffic pattern.

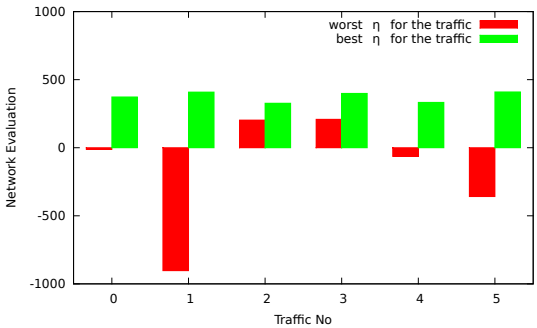


Figure 8. η-functions for different traffics in 16x16 RecMIN

This paper proposes to monitor the traffic flow by changes of queue lengths in the NoC buffers, in order to solve the problem of instability. Assume, each traffic corresponds to a vector $\vec{\theta}$. Thus, change in the traffic flow is observed by $\vec{\Delta\theta}$, the difference between two previous calculated $\vec{\theta}$-vectors:

$$\vec{\Delta\theta} = \vec{\theta_2} - \vec{\theta_1} = \begin{pmatrix} \theta_{0,2} \\ \theta_{1,2} \\ \vdots \\ \theta_{N-1,2} \end{pmatrix} - \begin{pmatrix} \theta_{0,1} \\ \theta_{1,1} \\ \vdots \\ \theta_{N-1,1} \end{pmatrix} \quad (3)$$

where $\theta_{i,j}$ is the length of the queue in the buffer j caused by traffic number i. In the more general case the system records the combination of topologies and corresponding traffic vectors in memory registers. Then, the reconfiguration algorithm can immediately change the NoC to the optimal topology, if the network traffic pattern repeats after some time.

B. The η-Algorithm

The η-algorithm uses the η-function for the evaluation and improvement of the network effectiveness. The algorithm receives the mean values for the network settings (throughput and delay) every 1000 cycles (number of cycles can be changed by the network designer). It calculates the value of η based on these means. If η-value falls below the specified threshold, the algorithm starts the reconfiguration.

Looking for an alternative structure is a typical global optimization problem of locating a good approximation to the global optimum of a given function. We used an exhaustive search of all possible topology reconfigurations, to find the optimal one. It is a reasonable alternative for small networks. (We used 16×16 RecMIN, where only $2^4 - 1$=15 reconfigurations are possible (the original configuration is not a reconfiguration). For networks with the higher number of RecCells, we recommend the usage of simulated annealing, genetic algorithms or other heuristic algorithms). Once all possible topologies for RecMIN have been iterated, the algorithm chooses the one with the maximum η-value.

The η-algorithm written in pseudo-code is shown below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_η:= calculate η;
BEGIN
 IF η<η_threshold THEN
    IF no reconfiguration is running THEN
     FOR i:=0
       TO i<all_possible_reconfigurations - 1
       DO
             simulate topologyi;
             calculate η;
            IF calculated η>best_calculated_η
            THEN
             best_calculated_η:=calculated_η;
```

```
            best_topology:=sim_topology;
           END IF;
     END FOR;
    END IF;
 END IF;
 RETURN best_topology;
END;
```

Advantages and Disadvantages of the η-Algorithm: The main advantage of the η-algorithm is the possibility of finding the optimal network topology for any traffic. Fig. 8 shows the analysis of six NoC traffics (in 16×16 RecMIN consisting of four RecCells) using the η-algorithm. The chosen priority weights are $C_{\varsigma_i} = C_{\tau_i} = 50$ [clock cycles/flit] and $C_{\delta_i} = -1$ [clock cycles]$^{-1}$ for all $i \in \{0, .., N-1\}$. For each traffic, Fig. 8 shows two values: the minimum and maximum value of η, which can be achieved by reconfiguring the network with the η-algorithm.

Fig. 8 shows the result of six chosen traffics given by the η-algorithm. It can be seen that for some traffic flows (e.g., traffic nos. 1 and 5) it is possible to achieve a good improvement of network performance by reconfiguration. On the other hand, some traffics exist (e.g., traffic nos. 2 and 3), for which reconfiguration does not lead to distinct enhancements. Therefor, the usage of the η-algorithm with exhaustive search is not reasonable for this kinds of traffics.

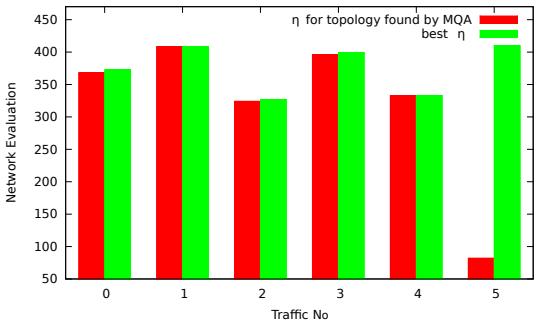


Figure 9. Comparison between the MQA and the η-algorithm for different traffics in 16x16 RecMIN

The additional disadvantages of the η-algorithm are:

1) The algorithm requires constant conduct of statistics of throughput and delay for the sources and targets in NoC. More sophisticated IP-cores (responsible for the collection of statistical data) have to be integrated in the network interfaces, to accomplish this task. This increases the chip area occupied by the network. Accordingly, the entire SoC production cost increases.
2) The algorithm deals with a large search space, when dealing with big NoCs consisting of many RecCells. This problem can be solved by using, e.g., simulated annealing. However, there is no guaranty of finding the optimal solution by the η-algorithm.

The η-algorithm is not very suitable for implementation in SoC, due to the disadvantage 1. However, it can be used in simulations. The designer can evaluate the effectiveness of other reconfiguration algorithms, comparing their results with the η-algorithm outcome.

C. The Minimal Queues Algorithm

Analysis of the various NoCs shows that the more effectively the network works, the shorter are the queues in the network buffers. Bottlenecks cause the queues in buffers on the respective network sections to rise. Subsequently, this effect generally leads to an increase of the length of the buffer queues in the entire network.

The idea of Minimal Queues Algorithm (MQA) is to react on increases of the lengths of the buffer queues in the network, and thereafter minimize these using reconfigurations. Observing the length of the buffer queues in a real SoC is much easier than keeping statistics of throughput and delay for sources and targets. Thus, the MQA is more suitable for implementation in SoC than the η-algorithm.

The trigger condition for the MQA is that the total number of packets in the network buffers exceeds some threshold specified by the developer. After that the MQA performs k reconfiguration steps. In each step, the MQA looks for switching the mode of one single RecCell that clearly shortens the lengths of the buffer queues in the entire NoC. Thereby, the MQA begins at the RecCell with the longest buffer queues. (The number k is specified by the developer. We set k equal to half of the amount of RecCells used in a network, i. e., if a network consists of four RecCells $k = 2$). The reconfiguration process requires neither to stop the operation of the NoC nor to release it entirely from packets, according to technique shown in [9].

The MQA written in pseudo-code is given below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_buffer_sum:= calculate(buffer_sum)
BEGIN
 IF buffer_sum>buffer_sum_threshold THEN
    IF no reconfiguration is running THEN
       FOR i:=0 TO i<k - 1 DO
        list_of_tried_cells:={};
        FOR each RecCell DO
            switching_cell:= search for
               RecCell with
               the highest  buffer_sum_in_cell;
            IF switching_cell
                 ∉ list_of_tried_cells THEN
             switch RecCell mode  (swisching_cell);
             simulate topology;
             calculate(buffer_sum);
      #if the buffer queues does not decrease
            IF NOT (calculated buffer_sum <<
                 best_calculated_buffer_sum)
            THEN
                step back to previous topology;
                add  switching_cell  to
                          list_of_tried_cells;
            END IF;
        END FOR;
        END FOR;
    END IF;
 END IF;
 RETURN actual_topology;
END;
```

Advantages and Disadvantages of MQA: As mentioned, the MQA is more suitable for real SoC than the η-algorithm, because it uses information of buffer occupation, instead of throughput and packet delay values. Furthermore, the MQA does not use an exhaustive search of all possible reconfigurations. In worst case $k * N$ reconfiguration steps have to be performed.

The main disadvantage of the MQA is that it does not provide the optimal solution. (The MQA is an empirical algorithm). Fig. 9 presents the comparison of the network performance between the η-algorithm and the MQA. Only in one of six cases, the MQA did not find the global, but the local optimum (traffic 5).

D. The Pattern Identification Algorithm

Generally, the occurrence of a bottleneck in RecCell can be identified by the occupation of its buffers. So, if such a situation arises in a particular RecCell, the overloaded channel can be diagnosed by a pattern of the buffer queues in the RecCell. Accordingly, if one of those patterns is recognised during an operation of the NoC, the RecCell has to be reconfigured.

A Pattern Identification Algorithm (PIA) can be implemented. It monitors bottleneck occurrence by recognition of buffer occupation patterns in each RecCell and performs the required reconfiguration. If more than one pattern is identified, the PIA gives priority to RecCells according to the distance of their position to the targets.

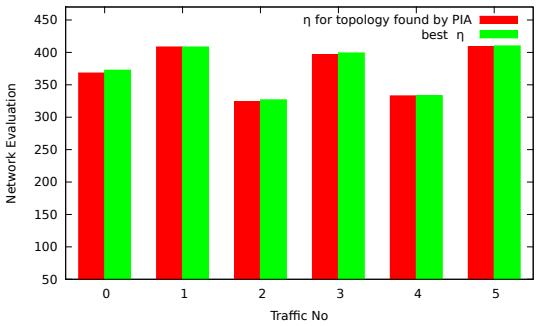


Figure 10. Comparison between PIA and η-algorithm for different traffics in 16x16 RecMIN

The PIA written in pseudo-code is given below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

BEGIN
 FOR i:=stage_number -1 DOWNTO 0 DO
   FOR each stage in RecMIN
             beginning from stage[i]  DO
     FOR each RecCell in this stage DO
      IF no reconfiguration is running THEN
        IF  is one of the patterns found  THEN
         reconfigure the cell according to the found pattern
        END IF;
      END IF;
     END FOR;
   END FOR;

 END FOR;

 RETURN actual_topology;
END;
```

Advantages and Disadvantages of PIA: An important advantage of the algorithm is that it does not search for a new topology by traversation of possible solutions. The PIA performs a reconfiguration only if it clearly improves the network efficiency. In worst case the PIA would do $2^n * k$ (where n is the index of stages in RecMIN, and k is the number of RecCells in each RecCell). But, in normal cases, the PIA is more efficient than the MQA comparing the number of reconfiguration steps.

Furthermore, a pattern search algorithm like the PIA uses buffer states as trigger information. This makes the implementation of the PIA in SoC simple. Also, the PIA performs reconfiguration steps for RecCells of the same RecMIN stage simultaneously so increasing the speed of the reconfiguration process.

The disadvantage of the PIA is that it requires implementation of additional memory registers in order to store the patterns in the NoC. Also, in case of miscarrying implementation of patterns, the RecMIN can become instable. Thus, the PIA will constantly detect one of the implemented patterns and fulfil infinite reconfiguration processes.

The PIA is the best of three algorithms proposed in this paper, for hardware realisation in SoC (in case that the patterns for the PIA are well implemented). So, for all of the six traffic flows that were used to test the performance of the three proposed algorithms, the PIA found an optimal NoC topology (Fig. 10).

V. Conclusion

In this paper, three reconfiguration algorithms were employed and evaluated, in order to benefit from the special capabilities of the Reconfigurable Multi-Interconnection Network (RecMIN) architecture. The η-algorithm, the minimal queues algorithm (MQA) and the pattern identification algorithm (PIA) allow the network to adapt itself to different traffic distributions. We evaluated the performance of the proposed reconfiguration algorithms with six chosen traffic flows and discussed the advantages and disadvantages of each algorithm. Finally, the η-algorithm is the best one for simulation. Therefore, the designer can evaluate the effectiveness of other reconfiguration algorithms, comparing their results with the η-algorithm outcome. However, the pattern identification algorithm is the most suitable reconfiguration algorithm for hardware realization in SoC.

References

[1] J. Owens, W. Dally, R. Ho, D. Jayasimha, S. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," Micro, IEEE, vol. 27, no. 5, Sept.-Oct. 2007, pp. 96 –108.

[2] D. Lüdtke, D. Tutsch, A. Walter, and G. Hommel, "Improved performance of bidirectional multistage interconnection networks by reconfiguration," in Proceedings of 2005 Design, Analysis, and Simulation of Distributed Systems (DASD 2005); San Diego. SCS, Apr. 2005, pp. 21–27.

 ISBN: 978-1-61208-341-4

[3] D. Lüdtke and D. Tutsch, “Lossless static vs. dynamic reconfiguration of interconnection networks in parallel and distributed computer systems,” in Proceedings of the 2007 Summer Computer Simulation Conference (SCSC’07); San Diego. SCS, Jun. 2007, pp. 717–724.

[4] ——, “The modeling power of CINSim: Performance evaluation of interconnection networks,” Computer Networks, vol. 53, no. 8, 2009, pp. 1274–1288.

[5] M. Al Faruque, T. Ebi, and J. Henkel, “ROAdNoC: Runtime observability for an adaptive network on chip architecture,” in Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on, Nov. 2008, pp. 543–548.

[6] ——, “Configurable links for runtime adaptive on-chip communication,” in Design, Automation Test in Europe Conference Exhibition, 2009. DATE ’09., april 2009, pp. 256 –261.

[7] A. Logvinenko and D. Tutsch, “A reconfiguration technique for area-efficient network-on-chip topologies,” in Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011 International Symposium on, June 2011, pp. 259 –264.

[8] ——, “Recsim - a simulator for reconfigurable network on chip topologies,” in Proceedings of the 26th European Simulation and Modelling Conference (ESM 2012). Essen, Germany, October 2012, pp. 144–151.

[9] A. Logvinenko, C. Gremzow, and D. Tutsch, “RecMIN: A reconfiguration architecture for network on chip,” in Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on, 2013, pp. 1–6.

[10] D. Tutsch, Performance Analysis of Network Architectures, 1st ed. Berlin: Springer Verlag, 2006.

[11] P. C. Wong and M. S. Yeung, “Design and analysis of a novel fast packet switch–pipeline banyan,” IEEE/ACM Transactions on Networking, vol. 3, no. 1, Feb. 1995, pp. 63–69.

[12] T.-Y. Huang and J.-L. C. Wu, “Alternate resolution strategy in multistage interconnection networks,” Parallel Computing, vol. 20, 1994, pp. 887–896.

[13] N. Boot, “Throughput and delay analysis for a single router in networks on chip,” Master’s thesis, Technische Universiteit Eindhoven, Netherlands, 2005.

 ISBN: 978-1-61208-341-4

Self-Adaptive Containers: Functionality Extensions and Further Case Study

Wei-Chih Huang and William J. Knottenbelt
Department of Computing, Imperial College London
{wei-chih.huang11, wjk}@imperial.ac.uk

***Abstract*—As the number of execution environments and application contexts rises exponentially, ever-changing non-functional requirements can lead to repeated code refactoring. In addition, scaling up software to support large input sizes may require major modification of code. To address these challenges, we have previously proposed a framework of self-adaptive containers which can automatically adjust their resource usage to meet Service Level Objectives and dynamically deploy the techniques of out-of-core storage and probabilistic data structures. A prototype with limited functionalities was implemented and applied to explicit state space exploration to prove the viability of our framework. In this paper, we broaden the library's functionalities through support for the important container class of key-value stores and integration of priority queues' functionalities into our previously-developed container class. We then utilise them in a new case study centred on route planning, adopting Dijkstra's shortest path algorithm. For this, a graph representing the full USA road network, which contains approximately 24 million nodes and 58 million arcs, is input to the algorithm so as to find the shortest paths from a random node to all the other nodes. The experimental results have shown that, under particular Service Level Objectives our library reduces update time by 21.4%, primary memory usage of node storage by 85.3%, and primary memory consumption required by the priority queue by 78%, compared with the Standard Template Library.**

***Keywords*-Self-Adaptive Systems; Containers; Standard Template Library; Probabilistic Data Structures.**

I. INTRODUCTION

Traditional software engineering methodologies are facing a new challenge – a rapidly growing number of system environments, where software is executed (e.g., tablets, servers, smartphones, laptops, routers). The applications may operate under different resource constraints and Quality of Service (QoS) requirements [1], based on the environments in which they are executed. Adapting software to each possible execution environment and application context in order to maintain QoS requirements is not a trivial job, especially in the situation where bursty and/or high-intensity workloads may frequently exhaust system resources [2] [3]. Further, this may take months or years of programmer effort to modify the majority of program code and may entail considerable programmer expertise [4][5]. These new challenges cannot be dealt with simply through use of traditional software engineering techniques [6–8], which results in either one of the two possible scenarios: a small code base which cannot guarantee QoS or multiple manually-optimised code bases which are difficult to maintain.

We have previously presented a framework of self-adaptive containers [9], which attempts to tackle the above-mentioned challenges via change of data structures. Instead of manually choosing a container and its underlying data structure, our self-adaptive containers provide two classes which automatically take such actions and dynamically change their underlying data structures in accordance with programmer-specified Service Level Objectives (SLOs) and the required functionalities. The former aims to easily satisfy ever-changing QoS requirements through modification of SLO specification, and the latter intends to provide a greater scope for efficiency optimisation. Conventional standardised container libraries are built for general purpose contexts, where all functionalities are always ready to be supplied, which restricts the possibility of optimisation. Through tighter functionality specification, our containers are able to exploit the techniques which can only be utilised when certain functionalities are entailed, including out-of-core storage and probabilistic data structures. To illustrate the viability of our framework, a prototype that fulfilled partial functionalities, highlighted in yellow in Figure 1, was implemented and utilised by the breadth first search algorithm, which explored up to 240 million states. The experimental results showed that our containers could not only dynamically boost their performance but save substantial memory space. Further, the containers' behaviour varied according to assigned SLOs, indicating that the self-adaptive containers could easily adapt to different execution environments.

Our framework with limited functionalities has been implemented to prove that it is feasible. In this paper, we add support for key-value stores (`IKeyValue`) and priority queues into our previously-built container class (`ICollection`). As described more fully in Sections IV and V, both of these data structures are widely-used in industry and are fundamental to many core computer science algorithms. `IKeyValue` supports commonly used member functions such as insert and the direct access operator. The functionalities of priority queues are supported by `ICollection`, which provides the required member functions and automatic deployment of out-of-core storage. The instances of either `ICollection` or `IKeyValue` can be assigned SLOs specified in the standard Web Service Level Agreement (WSLA) [10] format, which allows programmers to clearly and easily define resource constraints and QoS requirements. When currently-consumed resources violate the SLOs, our library's self-adaptive mechanism will determine if an adaptation action is needed in order to either satisfy the violated SLOs or reduce the degree to which the SLOs are contravened.

 ISBN: 978-1-61208-341-4

Our library is applied to a new case study, route planning, which adopts Dijkstra's algorithm, in order to show its enhanced functionalities. The experimental results suggest that our containers can effortlessly be adopted and considerably enhance both performance and memory efficiency. They also illustrate that the containers are capable of responding to programmer-specified SLOs.

This paper yields the following contributions:

- The functionalities of our self-adaptive container library are broadened through support for the fundamental container class of key-value stores and implementation of the functionalities of priority queues in our previously-developed container class, which expands application areas where our library may be applied.
- A new case study is investigated to illustrate our library's applicability. It shows how a naïve implementation of a core computer science algorithm in combination with our library can become resource-efficient and can achieve different programmer-specified Service Level Objectives.

The remainder of this paper is organised as follows. Section II introduces self-adaptive systems, reference models for building such systems and resource-aware systems. Section III describes our library's architecture and self-adaptive mechanism. While Section IV presents the design and implementation of key-value stores, Section V describes the out-of-core priority queue's implementation. Section VI presents the case study. Section VII concludes this paper and points out possibilities of future work.

II. Self-adaptive Systems, Reference Models, and Resource-aware systems

The foundation of the research regarding modern self-adaptive systems arose in the late 1990s and early 2000s, when IBM coined the term of autonomic computing [11], derived from human autonomic nervous systems, which could unconsciously control human bodies (e.g., heart rate, salivation, perspiration). A system adopting autonomic computing should involve the properties of self-configuration, self-healing, self-optimisation, and self-protection. To be equipped with these properties, a system should contain a self-adaptive cycle composed of an observation phase, an analysis phase, and an adaptation phase [12]. The observation phase is responsible for monitoring and collecting required data. The analysis phase determines if an adaptation action should be taken in accordance with the data reported from the observation phase and chooses a suitable adaptation action. The adaptation phase performes the adaptation action selected in the analysis phase.

After autonomic computing is introduced, a reference model for building self-adaptive systems, MAPE-K (monitor, analyse, plan, execute, and knowledge), is put forward [13] and implemented in several projects [14–16]. MAPE-K contains a cycle formed by the five functions, which are used to observe and collect data from managed resources, analyse the data, plan an adaptation action, perform the adaptation action to adjust managed resources, and store managed resources' goals, respectively. Garlan et al. [17] present another framework, Rainbow, which utilises an external approach for building self-adaptive systems.

To adapt to execution environments with different resource constraints, software should have the ability to detect its current resource usage. Sumatra [18], introduced by Acharya et al., is a Java extension, which provids four programming abstractions for monitoring resources and building resource-aware programs. In their work, they also suggest the awareness requirement, the agility requirement, and the authority requirement should be satisfied in the context of mobile agent software. However, the programmer overhead is relatively high in terms of adapting existing code to a tightly-specified mobile software architecture.

Among approaches automatically changing data structures to save resources is SILT [19] , which is a flash-based key-value store system featuring several underlying candidate data structures with data being converted between them according to the size of key fragments at run time. However, it only focuses on memory usage. Other QoS metrics (e.g., performance or reliability) are not taken into account. Indeed, there is no mechanism for sepcifying any Service Level Objectives, which leads to difficulties in adapting software to each execution environment and application context.

III. Library Architecture and Self-adaptive Mechanism

This section will briefly introduce our library's framework and self-adaptive mechanism. For full details, please refer to our previous publication [9]. As can be seen in Figure 1, the library consists of two major components, Application Programming Interface (API) and Self-Adaptive Unit (SAU). The API provides programmers with two template classes covering most functionalities of the Standard Template Library (STL) [20]: `ICollection`, which has been partially implemented in our previous prototype, and `IKeyValue`, supporting key-value stores. The member functions of `ICollection` and `IKeyValue` can be divided into operation interfaces and configuration interfaces. The former is a group of commonly-used operations. The latter acts as the means through which functionality requirements, SLOs, and the frequency with which the SLO compliance should be checked are imparted to the library.

The SAU, which performs operations and manages the self-adaptive mechanism, is composed of an Execution unit, a SLO store, an Observer, an Analyzer, and an Adaptor. The Execution unit performs container manipulation commands given by operation interfaces. The SLO store holds all SLOs laid down by configuration interfaces. The Observer monitors per operation response times, computes memory consumption, and calculates reliability when a probabilistic data structure is exploited. These operation profiles are then reported to the Analyzer, which determines whether an adaptation action should be taken. If an adaptation action is required, the Adaptor will be invoked to perform an adaptation action.

The self-adaptive mechanism of our library is a classical self-adaptive cycle [12], which is formed by the Observer, the Analyzer, and the Adaptor. The mechanism starts working when the Observer monitors the Execution unit to obtain

 ISBN: 978-1-61208-341-4

operation profiles (e.g., per operation response times, memory usage, and, where appropriate, reliability). The operation profiles are then sent to the Analyzer, which compares them with SLOs to determine if any SLOs are violated. If a certain SLO is violated, the Analyzer will decide if an adaptation action (e.g., the subdivision of the underlying data structure, the activation of out-of-core technique, or the deployment of probabilistic data structures) is required based on the following rules: (a) the adaptation action is expected to result in either the satisfaction of the SLO or a reduction in the degree to which the SLO is flouted and (b) the adaptation action is not expected to violate a currently-satisfied SLO of higher priority. We design these rules for two reasons. First, it may not be possible to meet all (or any) of the SLOs within resource constraints. Second, an adaptation action taken to address one violated SLO may cause the violation of another SLO. To solve these issues, each SLO is assigned a distinct priority according to its declaration sequence. The Analyzer addresses each SLO in priority order. If the SLO being addressed is satisfied, no adaptation is necessary. If the SLO is violated, the Adaptor is called in for an adaptation action.

IV. Key-value Stores Design and Implementation

Key-value stores, which represent data stored in pairs of keys and values, have been adopted in many industries managing large-scale data (e.g., Amazon [21], Facebook [22], Twitter [23][24], Linkedin [25]). As stored data accumulate, relational databases, which offer general purpose data stores, are incapable of providing acceptable data manipulation time. Many programming language data structures (e.g., map of the STL, HashMap of Java, and the dictionary data type in Python) and libraries (sparkey [26], LevelDB [27], YDB [28]) can be used to implement key-value stores and of course, recent years have seen the rise of persistent counterparts in the form of NoSQL databases (e.g., Cassandra [29], Riak [30], Tokyo Cabinet [31], Aerospike [32]). Our library supports the functionalities of key-value stores in `IKeyValue`, which chooses either a tree data structure, e.g. red black tree or AVL (Adelson-Velskii and Landis) tree or, where appropriate, a sparse Bloom filter [33], which transforms larger-sized elements into smaller-sized keys via hashing techniques to save considerable memory space, as the underlying data structure. As can be seen in Figure 1, `IKeyValue` API contains configuration interfaces and operation interfaces. The operation interfaces support the member functions which are needed to manipulate key-value stores, and the configuration interfaces including the constructor and `setAdaptationFrequency` are used for management purposes. The usage of `IKeyValue`'s constructor is illustrated as follows:

`IKeyValue`<K, V>(*op_desc*, *SLO_file*[, *freq*])

where *op_desc* specifies the required set of functionalities (so-called operation descriptors), *SLO_file* shows a path to an XML file containing a description of SLOs in WSLA format, and *freq* is an optional parameter defining the frequency with which the self-adaptive mechanism is activated.

`IKeyValue` is also capable of dynamically and automatically adjusting its underlying data structure through the SAU in order to meet SLOs. If its performance has to be improved, the underlying data structure will be subdivided. For example, when a sparse Bloom filter, which utilises a forest of AVL trees, is selected as the currently-used data structure, the number of AVL trees is increased to reduce the number of comparisons. If the reliability of `IKeyValue` needs to be increased, its underlying data structure will be subdivided as well. When the consumed memory space exceeds resource constraints, out-of-core storage may be activated. However, when the activation commences, some of the stored elements may be allocated to external memory, which makes the direct access operator (i.e., *operator[]*) unable to return a reference to the mapped value. To solve this issue, `IKeyValue`'s direct access operator will return a reference to a proxy class, which overloads the assignment operator (i.e., *operator=*) and the cast operator (i.e., *operator()*) to satisfy the functionalities of assignment and retrieval, respectively.

V. Out-of-core Priority Queue

Priority queues, whose underlying data structures are heaps, provide push operations as well as pop and top operations, which manipulate the largest (or smallest) element. As the number of stored elements increases, primary memory may be unable to store new elements, which leads to the utilisation of external memory. Because the performance of external memory is orders of magnitude slower than that of internal memory, out-of-core priority queues require I/O efficient algorithms [34–36]. In our library, the functionalities of priority queues specified by operation descriptors are embedded in `ICollection`, which now accepts a custom comparison operator as an optional template parameter, which defaults to less-than operator, to decide that either the largest or smallest element should be manipulated. When the internal memory limit has not been reached, `ICollection` behaves like the STL's *priority_queue*. Once the self-adaptive mechanism computes the memory consumption and sees it exceeds the primary memory limit, the mechanism will then take the following actions. First, it will sort the priority queue in primary memory and move the sorted elements to external memory. Next, the priority queue's largest (or smallest) element is inserted into a max (or min) heap which is intended to reduce response times of pop and top operations. These actions may be performed many times to keep memory consumption lower than the primary memory limit. When out-of-core storage is activated, pop and top operations should access not only the priority queue in internal memory but the root of the heap. If the root of the heap has to be removed, it will be deleted and the next larger (or smaller) element is then inserted into the heap.

Some researchers have proposed implementations of out-of-core priority queues (e.g., Standard Template Library for Extra Large Data Sets [37]), which considerably enhance I/O efficiency. However, they cannot dynamically determine when to trigger out-of-core storage, which deteriorates the performance of priority queues when in-core memory is sufficient. We have seen this drawback and aimed for our library to act as a controller which decides when to make use of out-of-core priority queues.

VI. Case Study

The case study chosen to illustrate our library's applicability is Dijkstra's shortest path algorithm, which has been extensively applied to route planning [38] and social network

 ISBN: 978-1-61208-341-4

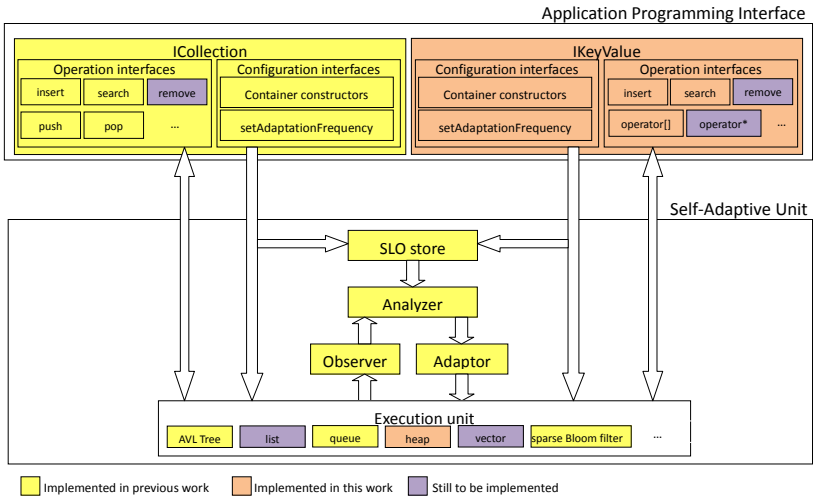


Figure 1. The highlight components of the library

analysis [39]. A naïve implementation of this algorithm is shown in Figure 2. Figure 3 displays the same algorithm via use of our library. As can be seen, the only difference between the two programs is the declaration of the key-value store variable, `Distance` (which stores the shortest distances from a given random node to all the other nodes), and of priority queue variable, `PQ` (which is used to locate the node with the shortest distance). To observe the library's behaviour under different SLOs, the following SLOs were assigned to `Distance`:

1) 80% of insertion times should be less than 1350 ns, and 90% of search times should be less than 500 ns.
2) Reliability should be higher than 0.995.
3) Memory use should be no more than 500 MB.

The above-mentioned SLOs were stored in *DistanceSLOs.xml* in WSLA format. An example of how to express SLOs fit for our self-adaptive containers in WSLA format is shown in [9]. Similarly, an SLO that requires the primary memory consumption of `PQ` to remain below 300 KB was assigned to *PQSLO.xml*. For `Distance`, the value of the optional parameter, `AdaptationFrequency`, was 100. In other words, the Analyzer was triggered every 100 operations. Naturally, the value of `AdaptationFrequency` may affect response times. In our previous case study, we have assigned different values to see the influence, which shows that when values of `AdaptationFrequency` are small, response times are lessened on account of a reduction in the Analyzer's activation times. As values rise, response times begin to increase due to delay of adaptation actions.

In this case study, a graph depicting the full USA road network [40], which contains approximately 23 million vertices and 58 million edges, was input. The performance and memory consumption from a given random node to all the others were compared, using the STL's class and our library. The SLOs of `Distance` were then input to different sequences to observe the library's behaviour. Finally, the memory consumed by both the STL's *priority_queue* and our library were displayed, showing improvement of memory efficiency.

A. Comparison with STL's map

To evaluate our library's effectiveness, Dijkstra's shortest path algorithm utilised to compute the shortest paths from a random node to all the other nodes was executed using the STL's map and our library. Figures 4 and 5 display average insertion and update times for `Distance`. The insertion time consumed by our library was close to that consumed by the STL's map. That was because our library spent extra time performing adaptation actions when elements were inserted. The sudden rises in insertion times indicated that our library changed its underlying data structures to boost performance or reliability. Although adaptation actions initially added to insertion times, they considerably improve scalability going forward. Indeed, insertion time and update time SLO are both subsequently maintained with only occasional adaptations.

Figure 6 depicts the memory space consumed by the STL's map and our library. Our library used an order of magnitude less memory space than the STL's map.

B. Influence of SLO priority

Figures 7 and 8 illustrate the performance-related SLOs and the time spent by our library under different priority orderings. For example, PerRelMem means that performance is the highest in order of priority, reliability has the next highest priority, and memory consumption's priority is the lowest. These figures indicate that when the given SLOs specified performance has higher priority over memory consumption, our library expends considerably less insertion time and update time. This phenomenon can be seen in the following orders of SLO metrics: PerMemRel, PerRelMem, and RelPerMem. When their performances cannot achieve the performance-related SLOs, the library still improves performance even if it means violating the memory limit. By contrast, when the memory-related SLO is the highest in order of priority, it causes our library to consume substantial insertion time and update time due to frequent out-of-core memory access.

The library's memory consumption conditions under the six priority sequences are depicted in Figure 9, which shows two different types of behaviour in accordance with priority in

 ISBN: 978-1-61208-341-4

```
void Dijkstra_algorithm(Graph G, Node s)
{
    priority_queue< pair<Node, double>,   compare > PQ;
    map<Node, double> Distance;
    Node u, v ;
    double cost;


    for (Node *w = G.start_node() ; w != G.end_node() ; w = G.next_node()) {
        Distance.insert(pair<Node, double>(*w, numeric_limits<double>::infinity()));
    }

    Distance[s] = 0;
    PQ.push(pair<Node, double>(s, Distance[s]));

    while (!PQ.empty()) {
        u = PQ.top().first;
        PQ.pop();
        pair<Node, double> *z = G.first_edge(u);

        for (; z ; z = G.next_edge(u)) {
            v = (*z).first ;
            cost = (*z).second;
            if (Distance[v] > Distance[u]+cost) {
                Distance[v] = Distance[u] + cost ;
                PQ.push(pair<Node, double>(v, Distance[v]));
            }
        }
    }
}
```

Figure 2. The naïve shortest-path algorithm

```
void Dijkstra_algorithm(Graph G, Node s)
{
    ICollection< pair<Node, double>,   compare > PQ(OP_PQUEUE, "PQSLO.xml");
    IKeyValue<Node, double> Distance(OP_INSERT|OP_INDEX, "DistanceSLO.xml", 100);
    Node u, v ;
    double cost;


    for (Node *w = G.start_node() ; w != G.end_node() ; w = G.next_node()) {
        Distance.insert(pair<Node, double>(*w, numeric_limits<double>::infinity()));
    }

    Distance[s] = 0;
    PQ.push(pair<Node, double>(s, Distance[s]));

    while (!PQ.empty()) {
        u = PQ.top().first;
        PQ.pop();
        pair<Node, double> *z = G.first_edge(u);

        for (; z ; z = G.next_edge(u)) {
            v = (*z).first ;
            cost = (*z).second;
            if (Distance[v] > Distance[u]+cost) {
                Distance[v] = Distance[u] + cost ;
                PQ.push(pair<Node, double>(v, Distance[v]));
            }
        }
    }
}
```

Figure 3. The resource-aware algorithm using self-adaptive containers

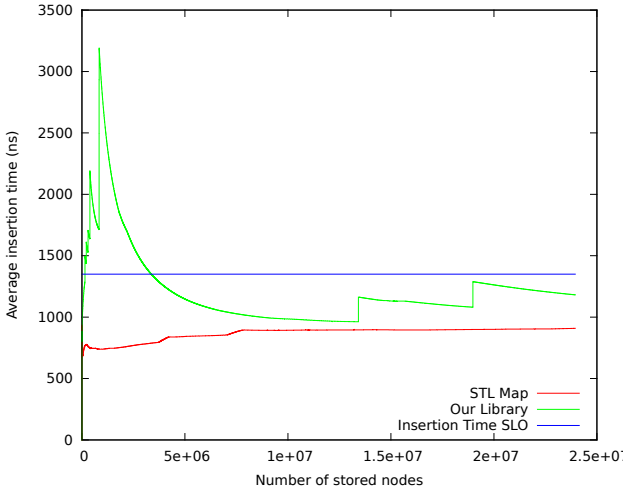


Figure 4. Average insertion time

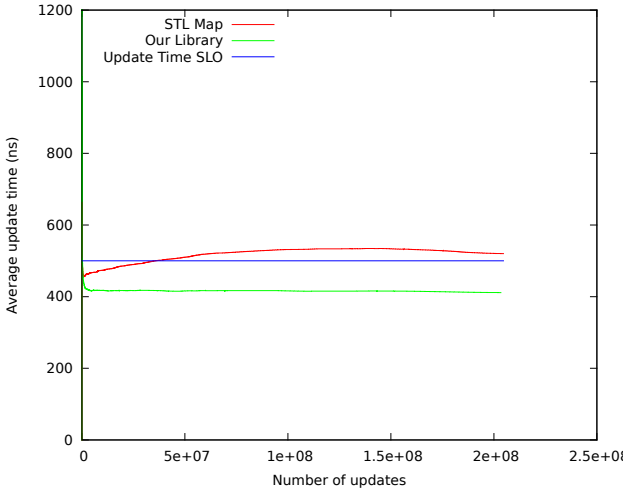


Figure 5. Average update time

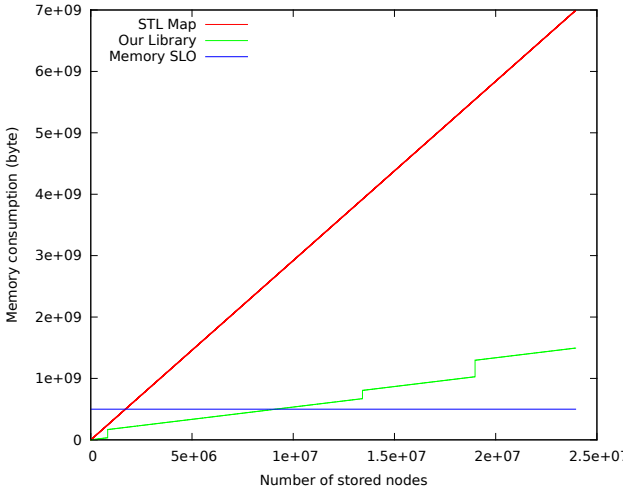


Figure 6. Memory consumption

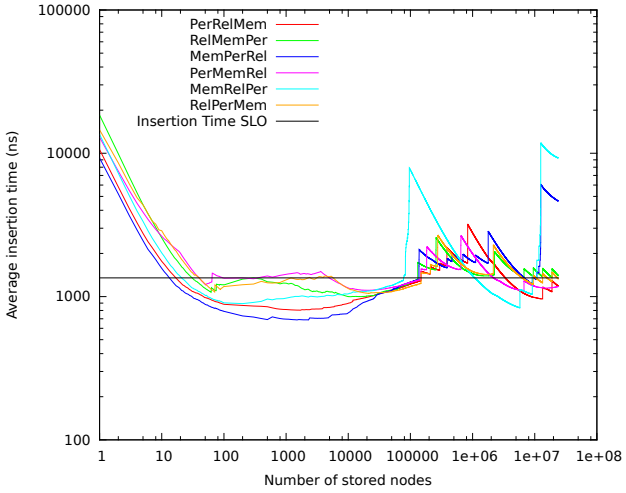


Figure 7. Average insertion times under different SLO priorities

 ISBN: 978-1-61208-341-4

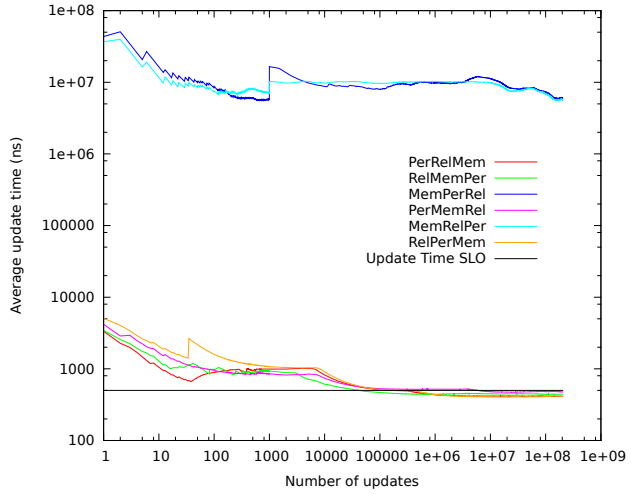


Figure 8. Average update times under different SLO priorities

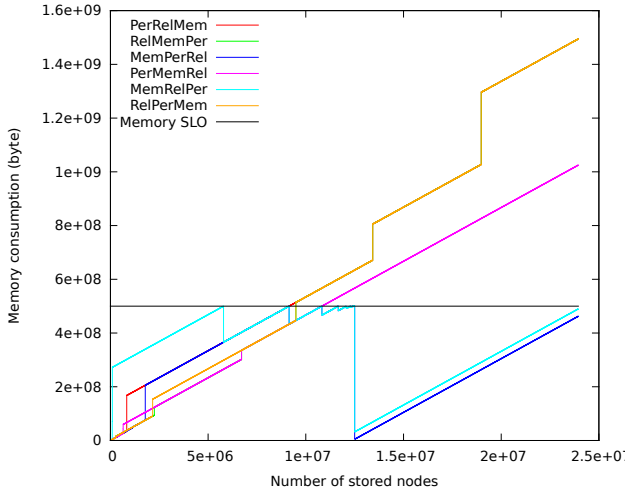


Figure 9. Memory consumption under different SLO priorities

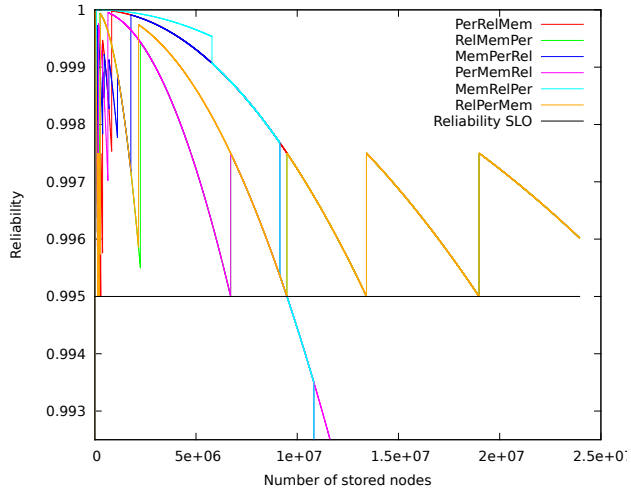


Figure 10. Reliability under different SLO priorities

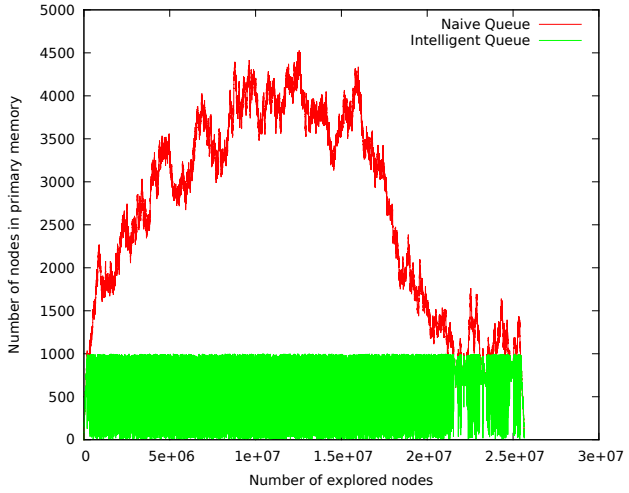


Figure 11. Memory consumptions of the naïve priority queue and the intelligent queue

memory consumption. When memory consumption is lowest in order of priority, more memory space is consumed to enhance performance or reliability. By contrast, when memory consumption has the highest priority (MemPerRel and MemRelPer), the consumed memory space is the least. This figure also indicates that when MemPerRel and MemRelPer reach the memory limit, our library, whose currently-used data structure is an improved sparse Bloom filter, reduces the number of AVL trees to save memory space. Once the number of AVL trees cannot be reduced, out-of-core storage is activated.

The variation in our library's reliability is depicted in Figure 10. As can be seen, when reliability has the highest priority (RelPerMem and RelMemPer), the reliability is kept at a desirable level – over 0.995. According to the two rules of our self-adaptive mechanism, when reliability is the highest in order of priority, it can be boosted without consideration of the side effects – most notably the increase in memory consumption. As a result, RelPerMem and RelMemPer rebound several times when reliability is equal to or lower than 0.995. But when reliability is lower in order of priority, the library's reliability descends as the number of inserted elements increases. Take MemRelPer for example. Memory consumption has higher priority than reliability, which implies that reliability cannot be improved once the memory limit is reached. Further, the reliability sharply deteriorates after adaptation actions which reduce the number of AVL trees are taken. While PerRelMem keeps reliability over 0.995, PerMemRel does not enhance reliability when the number of inserted elements is approximately one million. That is because for PerMemRel's memory consumption has higher priority than reliability. Consequently, when the current reliability is lower than the desired reliability, PerMemRel does not enhance reliability so as to prevent consumed memory exceeding the memory quota.

C. Out-of-core Storage for Priority Queue

As mentioned, the memory limit of the variable, `PQ`, was 300 KB. The primary memory consumptions using our library and the STL's *priority_queue* are shown in Figure 11. It indicates that our library consumed 300 KB, which was

a mere 78% of the memory space consumed by the STL's *priority_queue*.

VII. Conclusion and Future Work

In this paper, we have broadened the functionalities of our self-adaptive container library, which now supports the fundamental container class of key-value stores and enhances our previously-developed container class with the functionalities of priority queues. The new functionalities can dynamically exploit probabilistic data structures and out-of-core storage in an effort to meet different QoS requirements. Their efficacy has been proven in a case study, which illustrates how a naïve implementation of an algorithm utilising our library becomes scalable and resource-efficient by swift library-driven adaptations which successfully maintain programmer-specified Service Level Objectives in order of priority. Simultaneously, programmer overhead is kept low in terms of adapting software to a new environment. This can be easily achieved by means of redefining SLOs which are suitable for the resource constraints of a new environment.

So far, the library's self-adaptive mechanism follows a strict order of priority, which can be further extended to multi-objective optimisation methods such as a weighted product or a weighted sum of multiple SLOs. Another future direction of the library entails the cooperation with other container frameworks. Integrating them into our library can increase flexibility of the library in terms of its ability to meet SLOs in resource-constraint environments.

References

[1] A. Hervieu, B. Baudry, and A. Gotlieb, "Managing execution environment variability during software testing: an industrial experience," in Proceedings of the International Conference on Testing Software and Systems (ICTSS), 2012, pp. 24–38.

[2] D. Perez-Palacin, J. Merseguer, and R. Mirandola, "Analysis of bursty workload-aware self-adaptive systems," in Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ser. ICPE '12, 2012, pp. 75–84.

[3] I. Boutsis and V. Kalogeraki, "Radar: Adaptive rate allocation in distributed stream processing systems under bursty workloads," in SRDS, October 2012, pp. 285–290.

[4] R. Weiss, K. Krogmann, Z. Durdik, J. Stammel, B. Klatt, and H. Koziolek, "Sustainability guidelines for long-living software systems," in Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM), ser. ICSM '12, September 2012, pp. 517–526.

[5] J. Greenfield and K. Short, Software Factories: Assembling Applications with Patterns, Frameworks, Models and Tools. John Wiley and Sons, 2002.

[6] A. Mili, R. Mili, and R. Mittermeir, "A survey of software reuse libraries," Annals Software Eng., vol. 5, 1998, pp. 349–414.

[7] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

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

[9] W.-C. Huang and W. J. Knottenbelt, "Self-adaptive containers: Building resource-efficient applications with low programmer overhead," in Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2013, pp. 123–132.

[10] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," Journal of Network and Systems Management, vol. 11, 2003, pp. 57–81.

[11] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," 2011, presented at AGENDA 2001, Socttsdale, Available via http://www.research.ibm.com/autonomic/.

[12] M. Rohr et al., "A classification scheme for self-adaptation research," in Proc. International Conference on Self-Organization and Autonomous Systems In Computing and Communications (SOAS'2006), September 2006, p. 5.

[13] IBM Corp., An architectural blueprint for autonomic computing. IBM Corp., Oct. 2004.

[14] IBM. Autonomic computing toolkit. [Online]. Available: http://www.ibm.com/developerworks/autonomic/r3/overview.html [retrieved: April, 2005]

[15] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills, and Y. Diao, "Able: A toolkit for building multiagent autonomic systems," IBM Syst. J., vol. 41, no. 3, Jul. 2002, pp. 350–371.

[16] G. E. Kaiser, J. J. Parekh, P. Gross, and G. Valetto, "Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems," in Active Middleware Services, 2003, pp. 22–31.

[17] D. Garlan, S.-W. Cheng, A.-C. Huang, B. R. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," IEEE Computer, vol. 37, no. 10, 2004, pp. 46–54.

[18] A. Acharya, M. Ranganathan, and J. Saltz, "Sumatra: A language for resource-aware mobile programs," in Mobile Object Systems. Springer-Verlag, 1997, pp. 111–130.

[19] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, "Silt: A memory-efficient, high-performance key-value store," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ser. SOSP '11, 2011, pp. 1–13.

[20] D. R. Musser, G. J. Derge, and A. Saini, STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. Boston, Mass. Addison-Wesley, 2001.

[21] G. DeCandia et al., "Dynamo: Amazon's highly available key-value store," in Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, 2007, pp. 205–220.

[22] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, 2012, pp. 53–64.

[23] B. Fitzpatrick, "Distributed caching with memcached," Linux J., no. 124, Aug. 2004, p. 5.

[24] J. Petrovic, "Using memcached for data distribution in industrial environment." in ICONS, 2008, pp. 368–372.

[25] Linkedin. Project Voldemort. [Online]. Available: http://www.project-voldemort.com/voldemort/ [retrieved: January, 2014]

[26] M. Bruggmann. Sparkey. [Online]. Available: https://github.com/spotify/sparkey-java [retrieved: March, 2014]

[27] Google. leveldb. [Online]. Available: http://code.google.com/p/leveldb/ [retrieved: December, 2013]

[28] M. Majkowski. Ydb. [Online]. Available: http://code.google.com/p/ydb/ [retrieved: October, 2010]

[29] Cassandra. Apache Cassandra. [Online]. Available: http://cassandra.apache.org/ [retrieved: February, 2014]

[30] Basho. Riak. [Online]. Available: http://basho.com/riak/ [retrieved: February, 2014]

[31] F. Labs. Tokyo Cabinet. [Online]. Available: http://fallabs.com/tokyocabinet/ [retrieved: August, 2012]

[32] Aerospike. Aerospike. [Online]. Available: http://www.aerospike.com/ [retrieved: February, 2014]

[33] W. Knottenbelt, "Performance analysis of large Markov models," Ph.D. dissertation, Imperial College of Science, Technology and Medicine, February 2000.

[34] Chiang et al., "External-memory graph algorithms," in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 139–149.

[35] U. Meyer, P. Sanders, and J. F. Sibeyn, Eds., Algorithms for Memory Hierarchies, Advanced Lectures [Dagstuhl Research Seminar, March 10-14, 2002], ser. Lecture Notes in Computer Science, vol. 2625. Springer, 2003.

[36] N. R. Zeh, "I/O-efficient algorithms for shortest path related problems," Ph.D. dissertation, Carleton University, April 2002.

[37] R. Dementiev, L. Kettner, and P. Sanders, "STXXL: Standard Template Library for XXL data sets," Software: Practice and Experience, Aug 2007.

[38] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," in Algorithmics of Large and Complex Networks. Lecture Notes in Computer Science. Springer, 2009.

[39] U. Brandes, "A faster algorithm for betweenness centrality," Journal of Mathematical Sociology, vol. 25, 2001, pp. 163–177.

[40] 9th DIMACS Implementation Challenge. Shortest paths. [Online]. Available: http://www.dis.uniroma1.it/challenge9/download.shtml [retrieved: June, 2010]

 ISBN: 978-1-61208-341-4

An Adaptive Approach to Self-Healing in an Intelligent Environment

Guanitta Brady, Roy Sterritt, George Wilkie
School of Computing and Mathematics
University of Ulster
Jordanstown, Northern Ireland
brady-g6@email.ulster.ac.uk, {r.sterritt, fg.wilkie}@ulster.ac.uk

***Abstract*— In this paper, we address the management of sensor faults in an intelligent environment. Our proposed approach aims to introduce self-healing as a method of fault management. This approach is based on the use of adaptive finite state machine automata which handle suspicious sensor behavior. These state machines communicate with a mobile robot which investigates the error states detected through the sensors in the environment in order to learn from the anomalies and adapt to the changes in sensor behaviors. Additionally, we have determined that two types of fault may arise: systemic faults which the system may learn from and adapt to, and random faults which the system may compensate for through the use of a mobile robot as a sensor substitute.**

Keywords-fault tolerance; self-healing; sensor substitution; intelligent environment.

I. INTRODUCTION

Since the introduction of the concept of intelligent environments, we have seen an increase in the applications of the sensor technologies synonymous with these environments. A promising application of those technologies is within the smart home for the delivery of pervasive care [1]. These environments aim to facilitate the monitoring of elderly occupants who suffer from cognitive impairments or degenerative conditions, such as dementia [2], and to support independent living [3]. In order for these environments to function effectively they must be tolerant of faults. The efficient functionality of sensor technologies in care homes for the elderly is crucial to ensuring the safety of the environments occupant. Those who suffer from dementia are often prone to wandering behavior [4]. As a consequence, there is great potential that those who leave their homes undetected may place themselves in danger [2]. For this reason, this research focuses on the monitoring of activity about a door in an intelligent environment. This work is motivated by the widespread instances of dementia patients who have left their care homes undetected [5] [6].

The prevalence of sensor technologies coupled with the increasing complexity of information systems is leading us to the need for systems that are capable of self-management and self-adaptivity [7]. Our approach aims to take the initial steps in introducing the first of four key properties of an autonomic system: self-healing. This research aims to achieve this through the introduction of sensor substitution and adaptivity to the sensor technology about a door in an intelligent environment in order to provide the self-healing and self-management of sensor faults and anomalous behavior respectively. The proposed approach makes use of multiple finite state machines coupled with the use of a fuzzy logic rule base and adaptive learning techniques in order to provide intelligent adaptive fault management. This is achieved through the systems own investigation of its error states from which the system may learn new behaviors and adapt its policies accordingly.

The remainder of this paper is organized as follows: Section II provides an overview of related work. In Section III we present our design. Section IV discusses some preliminary results. We conclude with Section V in which our future work is outlined.

II. RELATED WORK

A comprehensive summary of the use of finite state automata in the design of reliable software is presented by Wason et al. [8]. Whilst the use of finite state machines for the purpose of introducing fault tolerance is not a new concept [9], there exists a research gap in terms of the need to further explore the extent to which a system may be made autonomic through the use of state machines so that a system may investigate and learn from its error states [8], in order to create a stronger awareness of the conditions under which the system is expected to perform. In order to achieve this, a system must be both self-aware and environment aware [10].

A popular approach to the management of faults is the use of redundancy [11]. These approaches focus on the use of additional hardware as a fail over mechanism when their counterparts degrade. These sensors are capable of measuring identical or closely related values. This approach does not resolve the underlying fundamental problem that hardware is subject to failures and even redundant components have the potential to be subject to a fault or failure; particularly if their data is only incorporated into the monitoring process periodically. For this reason, we propose that the use of adaptable software to compensate for the shortcomings of hardware devices as their behavior degrades is a practical and cost-efficient approach. Indeed, the large volume of research that is undertaken in the area of robotics for pervasive care suggests that in the future robots will have a more prevalent role [12], particularly in care home environments. We can utilize these robots to not only assist in the delivery of pervasive care, but to also assist in ensuring fault tolerance in an intelligent environment by providing a mobile means of delivering sensor substitution and the investigation of anomalous sensor behavior at the point of need.

 ISBN: 978-1-61208-341-4

III. DESIGN

In this section, we will describe the design of our system in terms of both the hardware and software topologies.

A. Hardware Topology

In our previous work [13], we investigated the viability of using an ultrasonic array mounted on a mobile robot as a means of substituting for a radio-based door mounted contact sensor. From this work, we concluded that two door states, opened and closed, could reliably be determined by the mobile array. Previously, our topology consisted of a radio contact sensor, pressure mat and the mobile robot Pioneer 3-DX from Adept Mobile Robots [14]. We extended this to include an additional pressure mat in order to detect when a person had passed through the door threshold. Our hardware topology is depicted in Figure 1.

1) Static Sensors: Based on our observations of the sensor data generated by the static sensors, we determined that these sensors may be viewed as "black and white" sensors as their readings dictate one of two states; the radio door contact sensor can return a "door opened" or a "door closed" value. Similarly, the pressure mats can return a true or false binary value to denote their activation or dormancy.

2) Mobile Sensors: In contrast to the static sensors the mobile sensors with which the robot is equipped, which include an ultrasonic array and an infrared sensor, may be viewed as "grey" sensors. This stems from the fact that whilst the static sensors can provide a simplistic piece of information depicting that they are in one of two possible states, the mobile sensors require pre-processing in order to derive information from their data about their perception of the world.

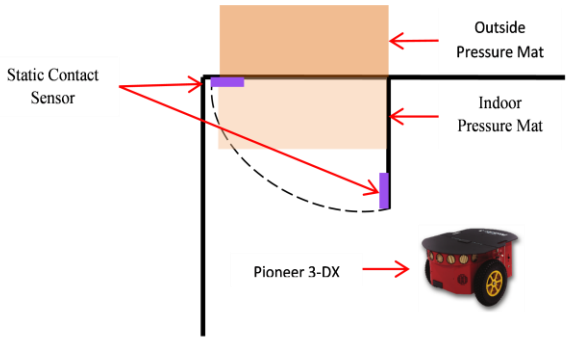


Figure 1. Hardware topology in experimental environment.

B. Software Topology

A high-level conceptual overview of the software system structure is presented in Figure 2. It is made up of two communicating finite state machines and two feedback loops. Each of the state machines communicates with the static sensors and mobile sensors respectively. It is through this continuous feedback that the states in the machines are driven. This is discussed in the following sub-section.

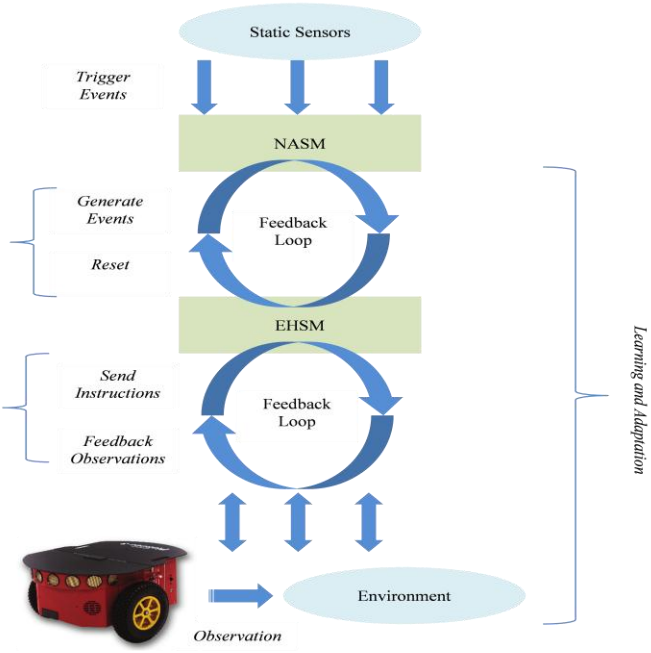


Figure 2. High-level overview of self-healing system structure.

1) Finite State Machines: We have designed two finite state machines based on our physical topology. It was determined that two state machines were required in order to allow for the concurrent monitoring of normal activity and the investigation of anomalous sensor behaviour. By designing the state machines to facilitate concurrency, effective monitoring can be delivered irrespective of the detection and investigation of anomalous behavior.

a) Normal Activity State Machine: The first finite state machine is the Normal Activity State Machine (NASM). This machine handles the expected pattern of static sensor activations about a door. It was determined that based upon the combinations of the static black and white sensors that eight possible states could exist, given that each sensor could be determined to be in one of two states, which denote normal activity about a door. The number of possible sensor events is: 2^3= 8 possibilities, where there are two possible events which may fire for each of the three sensors and no event may be repeated in the course of a normal traversal of a doorway. The states in the NASM are:

- S0 Door closed
- S1 Door opened
- S2 Person inside & closed
- S3 Person outside & closed
- S4 Person inside & outside & closed
- S5 Person inside & open
- S6 Person outside & open
- S7 Person inside & outside & open

The states S2, S3, S5 and S6 correspond to a single person approaching the door. The states S4 and S7 correspond to the possible presence of another person on the outside

 ISBN: 978-1-61208-341-4

pressure mat in addition to the presence of a person on the indoor pressure mat. These states are driven by seven events: six of these events are the static sensor events and the seventh is a reset event which may restore the state machine to a specified state. The Reset event may only be generated by the Error Handling State Machine (EHSM).

b) Error Handling State Machine: The second state machine; the EHSM, consists of nine states and ten events which drive those states. The states in this machine are derived from the combinations of the environment sensors which may exhibit anomalous behaviour. The events in this state machine stem from two sources. The first source is the NASM from which events are generated via its actions to the EHSM upon the receipt of anomalous sensor readings. The second source of the events in the EHSM is the mobile robot. Upon the detection of anomalous behaviour the mobile robot is deployed to the site of the sensor failure where its role is twofold: in the first instance the robot must deliver feedback of its sensor readings. This data is processed in order to provide a corresponding value for the sensor it is investigating so that normal monitoring of activity about a door can continue whilst the anomaly is being investigated. The results of this analysis are input to the EHSM through a feedback loop. This, in turn, generates an event into the NASM via a second feedback loop instructing it what state to transition into based on the robot's sensor readings. Secondly, the robot's sensor data is processed in parallel with the monitoring activity in order to identify patterns in changes in sensor behaviour. It is through this reflective analysis over time that adaptive learning is facilitated.

2) Anomaly Identification: From our observations of our static sensors' behaviour over time, we determined that two types of anomaly may be exhibited: random anomalies and systemic anomalies.

a) Random Anomalies: Random anomalies are defined as those which occur sporadically such as the absence of an expected sensor activation. These anomalies are addressed directly through the mobile robot, which positions itself at the door and provides substitution of the sensor in question. Whilst providing substitution, the robot feeds back its own sensor data. This data is then pre-processed and correlated with that of the static environment sensors for anomaly verification and fault diagnosis.

b) Systemic Anomalies: Systemic anomalies are defined as those which occur as sensor behaviors change over time. Our current research leads us to believe that these changes in behavior may be attributed to the degradation of hardware resulting in behaviors such as slower relay time, battery deterioration or the receipt of multiple sensor events for one real-world event. This requires further investigation in order to verify the validity of this hypothesis. It is these anomalies that the system must investigate fully in order to learn about the changes in the behavior of the sensors in the environment. To this end, adaptive learning [15] must be applied so that the system may adapt its policies.

IV. DISCUSSION

When an anomalous sensor reading is received by the NASM, a corresponding action, dependent on sensor type, inputs an event to the EHSM. A feedback loop operates between the NASM and EHSM whereby, upon successful investigation of the anomaly, the EHSM may then generate a Reset event into the NASM in order to restore its function. Alternatively, the EHSM may generate a new action into the NASM which corresponds to a systemic anomaly which has been detected in a given static sensor so that the NASM may handle the occurrence of that sensor behavior in future without reporting the event to the EHSM as a new anomaly.

By utilizing a feedback loop, actions may be dynamically generated into the NASM. These actions are the result of the investigation by a mobile robot and analysis by the system of the sensor behaviors. By providing the dynamic generation of actions into the NASM, the adaptivity of the state machines policies can be achieved. This approach has the potential to bring greater flexibility to the system and more robust fault tolerance without the need for human intervention. Therefore, the adaptivity in this system will be achieved through the NASM. This is facilitated by the feedback of the results of the robot's investigation of the environment sensors via the EHSM.

When the EHSM receives an event from the NASM, it will then trigger a transition to the relevant state dependent on the sensor or sensors that have been deemed suspicious. The EHSM then performs an action relevant to the anomalous sensor. Initially, the EHSM's action will instruct the robot to navigate to the site of the sensor. In order to do this, the robot requires a-priori knowledge of the environments structure. The method of navigation is not pertinent to this research piece as it is an area which is widely covered by roboticists. We are concerned only with the fact that the robot can consistently navigate to a pre-designated position, which is dictated by the unique identifier of the specific sensor in question, using a map of the environment and collision avoidance. Before the results of the robot's observations can be fed back to the EHSM, pre-processing and analysis is required. It is through this pre-processing and analysis that investigation of the anomalous behavior takes place. This, in turn, facilitates the systems learning about anomalies and changes in sensor behaviors.

The analysis of the data received from both the robot and the static sensors provides the system with the ability to begin to identify systemic anomalies. Subsequently, the system may then learn about its faults and adapt its policies to account for the new behaviors exhibited by the static sensors. For example, if the door contact sensor develops behavior whereby it fires three sensor events for one real-world door opened event instead of once, as would ordinarily be expected, from the correlation of the robot's observations with the static sensor events the system may then learn that this pattern is a systemic anomaly and accordingly adapt its

 ISBN: 978-1-61208-341-4

policy to allow three door opened events to occur, through the adaptation of actions in the NASM, before an anomalous contact sensor event is passed to the EHSM in future.

In the course of our research we have observed the door contact sensor which is part of our system topology. From these observations, we have determined that there are instances in which sensor events are not received. During our experiments, it became evident, for example, that the sensor signals were lost on occasion when a door was closed. However, this occurrence was not consistent. As a result it is difficult to adapt to this system behavior. Consequently, the mobile robot was instructed to navigate to the door site location. Once there the robot's role is two-fold in this instance: it must use its ultrasonic array to determine the door state and it must also verify the anomalous behavior. When the robot's data from its ultrasonic array is fed back it takes over the role of the contact sensor in the finite state machines, following pre-processing of the data, until the issue with the door contact can be verified as either a random or systemic anomaly.

It is evident from the above that the utilization of a mobile robot as a means of investigating and verifying anomalous sensor behavior has the potential to address two types of anomalous behavior in the sensors about a door. We have used anomalous readings from the door contact sensor by way of example here, however; our approach would also hold for the investigation of anomalous pressure mat readings. Whilst the implementation is incomplete, we believe that we can contribute to the adaptive management of sensor faults in an intelligent environment through our utilization of a mobile robot for investigation of the anomalous behavior coupled with the dynamic adaptation of system policies following the systems own investigation and analysis of its suspected error states.

V. CONCLUSION AND FUTURE WORK

In this paper, a proposed approach to the self-healing of door based sensors in an intelligent environment was presented. This research utilizes a mobile robot in order to provide sensor substitution and verification of anomalous sensor behavior. By utilizing a mobile robot to investigate its anomalies, the system may determine the nature of the anomalous behavior. We have devised that this behavior may be categorized into two types of anomalies based on observations of a mobile robot over time; systemic and random anomalies. It is through this process of investigation of its own anomalous events that the system may learn from the behavior of the sensors contained therein and adapt its policies accordingly.

The ideas presented in this paper are in the process of being implemented, and subsequently require further evaluation. The preliminary results are promising and appear to offer a useful means of introducing self-healing through sensor substitution and software adaptation into an intelligent environment in order to ensure the tolerance of static sensor faults.

Our future work will consist of experimentation and evaluation of our approach, following the completion of our implementation. The performance of the proposed approach will be qualitatively evaluated in order to establish a clear picture of the performance of our approach in the real world. Adaptive learning for the dynamic generation of system policies through the use of a fuzzy rule-base [16] will be further investigated in order to establish how well the system learns using this approach.

ACKNOWLEDGMENT

Guanitta Brady's PhD research is funded by DEL (http://www.delni.gov.uk) and has been awarded the 2012 Annual Research Bursary from HISI (http://www.hisi.ie/).

REFERENCES

[1] M. E. Pollack, "Intelligent Technology for an Aging Population: The use of AI to Assist Elders with Cognitive Impairment," AI Magazine, vol. 26, no. 2, pp. 9-24, Summer 2005, doi: 10.1609/aimag.v26i2.1810

[2] A. J. Bharucha et al., "Intelligent Assistive Technology Applications to Dementia Care: Current Capabilities, Limitations, and Future Challenges," The American J. of Geriatric Psychiatry, vol. 17, no. 2, pp. 88-104, Feb. 2009, doi: 10.1097/JGP.0b013e318187dde5.

[3] F. Sadri, "Ambient Intelligence: A Survey," ACM Computing Surveys (CSUR), vol. 43, pp. 36:1-36:66, Oct. 2011, doi: 10.1145/1978802.1978815.

[4] C. K. Y. Lai and D. G. Arthur, "Wandering Behaviour in People with Dementia," J. of Advanced Nursing, vol. 44, pp. 173-182, Oct. 2003, doi: 10.1046/j.1365-2648.2003.02781.x.

[5] S. Davies. (2013, November 4), Wife's anger after Southampton care home unaware dementia patient has walked out [Online]. Available: http://www.dailyecho.co.uk/news/10782832.Wifes_anger_over_care_home_blunder/

[6] CTV British Columbia. (2013, December 9), Family wants answers after senior leaves care home, dies outside [Online]. Available: http://bc.ctvnews.ca/family-wants-answers-after-senior-leaves-care-home-dies-outside-1.1582407

[7] P. Horn. *Autonomic Computing: IBM Perspective on the State of Information Technology* [Online]. Available from: http://www.research.ibm.com/autonomic 2001.10.15

[8] R. Wason, P. Ahmed, and M. Q. Rafiq, "Automata-Based Software Reliability Model: The Key to Reliable Software," Int. J. of Software Engineering & Its Applications, vol. 7, no. 6, pp. 111-126, Nov. 2013, doi: 10.14257/ijseia.2013.7.6.10.

[9] R. Leveugle and L. Martinez, "Design Methodology of FSMs with Intrinsic Fault Tolerance and Recovery Capabilities," Proc. Euro ASIC 1992 (AISC 92), IEEE Press, June 1992, pp. 201-206, doi: 10.1109/EUASIC.1992.228024.

[10] R. Sterritt and D. Bustard, "Towards an Autonomic Computing Environment," Proc. IEEE Int. Workshop on Database and Expert Systems Applications (DEXA 03), IEEE Press, Sept. 2003, pp. 694-698,, doi: 10.1109/DEXA.2003.1232103.

[11] V. P. Nelson, "Fault-Tolerant Computing: Fundamental Concepts," Computer, vol. 23, pp. 19-25, July 1990, doi: 10.1109/2.56849.

[12] A. M. Okamura, M. J. Matarić, and H. I. Christensen, "Medical and Health-Care Robotics," IEEE Robotics & Automation Magazine, vol. 17, pp. 26-37, Sept. 2010, doi: 10.1109/MRA.2010.937861.

[13] G. Brady, R. Sterritt, and F.G. Wilkie, "An Investigation into the Viability of a Mobile Ultrasonic Array as a Sensor Substitute in an Autonomic Intelligent Environment," Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC 13),

 ISBN: 978-1-61208-341-4

IEEE Press, Oct. 2013, pp. 577-582, doi: 10.1109/SMC.2013.104.

[14] Adept MobileRobots LLC., Pioneer 3 Operations Manual. Amherst, NH, 2010.

[15] R. M. Bahati, and M. A. Bauer, "An Adaptive Reinforcement Learning Approach to Policy-Driven Autonomic Management," Proc. IEEE Int. Conf. on Autonomic and Autonomous Systems (ICAS 09), IEEE Press, April 2009, pp. 135-141, doi: 10.1109/ICAS.2009.58.

[16] I. B. Türkşen, "A Review of Developments in Fuzzy System Models: Fuzzy Rule Bases to Fuzzy Functions," Scientia Iranica, vol. 18, no. 6, pp. 522-527, June 2011, doi: 10.1016/j.scient.2011.04.001.

 ISBN: 978-1-61208-341-4

OfficeMate: A Study of an Online Learning Dialog System for Mobile Assistive Robots

Steffen Müller, Sina Sprenger, Horst-Michael Gross
Ilmenau University of Technology
P.O. Box 10 05 65, 98684 Ilmenau, Germany
Email: Steffen.Mueller@tu-ilmenau.de,
Sina.Sprenger@gmx.de, Horst-Michael.Gross@tu-ilmenau.de

***Abstract*—Service robots in the near future are supposed to live together with humans in their private homes for a longer time period. In this situation, experience and attitudes of the users change and thus, the robot has to develop its behavior, too, and it has to adapt to the user's way of interaction and the user's needs. The contribution of this paper is a probabilistic decision planner implementing the idea of online learning dialog strategies for a mobile service robot in long-term interaction. The planning system is part of a modular multi-modal dialog system and allows for an autonomous personalization of the robot's actual interaction behaviors. A model of observed transitions and user's rewards using mixtures of discrete samples is proposed for efficient inference in a factor graph model. The practicability of the dialog system and the rewarding mechanism have been evaluated in a ten day realworld experiment with 16 users.**

Keywords–online learning; dialog system; probabilistic planner

I. INTRODUCTION

The work presented has been conducted in the scope of the research group SERvice RObotics for health (Gesundheits) Assistance (SERROGA) [1], which intends to develop demonstrators for robotic applications in the context of prevention and assistance for elderly people living alone in their home environment. As a vision, we see a robot, that is living together with the human in a long-term interaction situation. Furthermore, we expect the development of an emotional binding of the user to his or her personal robot over the time, which is reinforced by the ability of the system to adapt to the user's needs and preferences. The intended platform to be used in the SERROGA project is a Scitos-G3 (see Fig. 1), that has been developed in the EU funded project CompanionAble [2][3]. An intuitional communication is realized by a multi-modal user interface consisting of a touchscreen, touch sensitive cover, and a touch sensitive patch of fur used for petting the robot. Actual work in progress intends for inclusion of speech recognition as an additional input channel. For output, the robot can use synthesized voice, the screen, as well as an artificial face consisting of two eye displays.

A laser range finder and a Kinect sensor, together with a differential drive enable autonomous localization and navigation skills. A fish-eye camera is used for person detection and tracking, which is a key functionality for successful interaction. Additionally, for giving explicit feedback by a user, special positive and negative reward buttons are placed on top of the screen.

The aim of this work is to enable and understand a long-time development and adaptation of a multi-modal human-robot dialog. In that context, we identified three phases in the interaction. At the beginning, in the first phase, the user needs to get to know the robot. The system has to give advice how to use it and has to introduce its capabilities. The dialog initiative is primary at the side of the robot. Later, in a second phase, when the user knows better about the capabilities of the robot, the initiative will be in the user's hand, and the robot should learn the user's preferences and needs. That means, the robot is supposed to learn which services are used in which situation and what are the user's attitudes towards the various options the robot has in its dialog behavior. Also, preferred selections are learned by the robot in order to apply that knowledge in the third phase. When that third, stable phase is reached, the robot can make use of the observations in interaction with that specific user. So it is able to act proactively depending on the current situation. Then, a mixed initiative dialog should emerge even with the limited domain of the robot's services. Nevertheless, the ability to learn and change interaction behavior should not be limited to the initial phases. Changes of user's attitudes have to be tracked continuously and life-long.

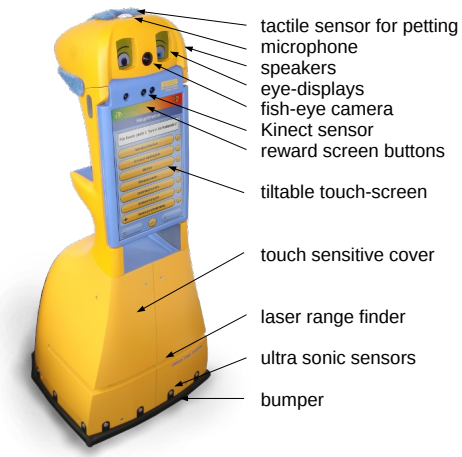


Figure 1. Scitos-G3 robot "Max" used for the evaluation.

The remaining part of this paper is structured as follows: First, a brief discussion of approaches from literature for learning dialog behavior is given, followed by the presentation of our own contribution. Thereto, in Section III, first, the dialog system is described at a glance, and afterwords, the adaptive parts, the probabilistic planner and the models used, are explained. In Section IV, our experience with the described

 ISBN: 978-1-61208-341-4

system in user test is presented, and results are discussed.

II. Related Work and Discussion

In literature, several approaches for dialog modeling can be found mainly in the field of speech-based dialog systems, which also have capabilities of adaptation. A common approach is using Reinforcement Learning techniques for optimization of a dialog flow, even when inputs are uncertain. The Partially Observable Markov Decision Process (POMDP) is used here, but this comes along with high computational effort, which is intended to be overcome by several approaches for simplification [4][5]. These approaches try to find a policy in order to maximize the discounted future reward, that mainly is generated by a system internal reward function. Therefore, the system designer has to define in advance what the long-term goals during upcoming dialogs will be.

Pineau et al. [6] applied a POMDP model for learning an optimal dialog behavior for a service robot called Pearl. Although they applied a hierarchical decomposition of the assistance task, the complexity of the realizable functionality is very limited. That approach also relies on a hand crafted reward function and the policy is computed offline before application. Thus, the robot can not consider individual user's characteristics discovered at runtime.

One disadvantage of many similar Reinforcement Learning-based approaches in the domain of dialog learning is the batch update, where in a training phase interaction data is acquired, and afterwards the optimization run is applied offline in order to generate the productive dialog system.

A model for learning a behaviour online from direct user rewards is called "Training an Agent Manually via Evaluative Reinforcement" (TAMER) [7][8]. This approach explicitely models, which feedback a user gives for a certain state-action pair, and then acts greedily in order to get the maximum reward for the next action. The argumentation for this λ_0 strategy is the idea, that the human supervisor estimates the utility value of a state-action pair and already represents this in the reward signal. This might be correct to a certain degree, but has a clear disadvantage. The system can only act in order to achieve user goals, but is not able to incorporate internal goals or wishful target states. One interesting aspect of the TAMER model is the reward model. This allows to predict the user rewards and apply them internally - even if the user is not giving feedback for each action. Thus, this model allows that the user has to get active by giving feedback only if s/he wants to modify the behavior, not if s/he is pleased with it.

An alternative dialog system applying probabilistic inference is presented in [9] and later in [10]. Inference techniques have been applied to a statistical model of the dialog in order to reason the goals the user might have in mind and, therefore, decide which information needs to be asked or given in the next steps of dialog. Unfortunately, this idea is not directly transferable to our scenario, where the goal of the dialog is not only determined by the user but also by the system itself (e.g., the robot should engage the user in communication or physical activities). Additionally, the required direct reward to be given by the user, which is used to modify the way things are communicated, is hard to introduce in that approach.

In our approach, we need to combine system internal goals, which mainly are i) fast task completion and ii) less correction steps by the user with explicit and implicit rewards given directly by the user during the interaction. As explicit reward, we consider positive or negative feedback by means of pushing the respective like or dislike buttons on the screen, while implicit rewards are unconscious signals like the rate of petting the robot, or simply ignoring the robot's attempts to interact with the user.

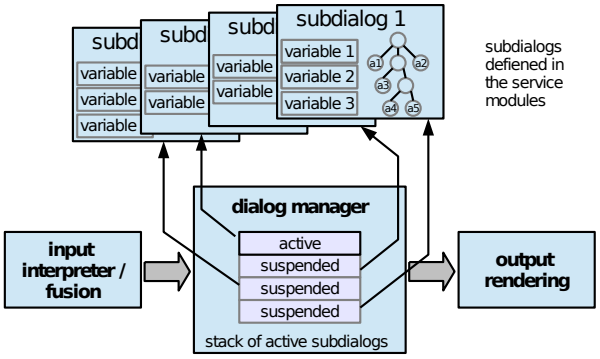


Figure 2. Architecture of our dialog system - The dialog manager holds a stack of subdialogs each defined by as set of state variables a decission tree of possible actions.

To get capability of online learning/adaptation, we would like to get rid of the complicated optimization problem of finding a complete policy each time we get new observations, although we are only interested in the optimal action for the current situation. This is possible by means of an online planning mechanism and also allows for changing optimization goals as well as discovering new states of the dialog at runtime, which would be difficult or even impossible for implicit planning methods.

Dialog modeling techniques existing today are mostly related to a very complex application development process. Our aim is to provide a framework for rapid application development, which is realized by combining simple frame-based multi-modal dialog with the capabilities of optional adaptation without introducing additional configuration effort. A key to a manageable design effort is the possibility for problem decomposition. According to hierarchical abstract machines [11], that are also used in the Reinforcement Learning domain for restriction of the action space, in our approach, individual subdialogs are defined as independent modules, each restricting the policy to a reasonable subset and having the ability of calling other subdialogs on demand. How this modularization is realized in our system is explained in the next section.

III. Adaptive Modular Dialog Systems

The implementation of the dialog system is based on the robot middle-ware MIRA [12][13] and implements the control layer in our software architecture [2]. Therefore, it integrates well with the other robot software components for navigation, perception skills, and graphical user front-end and realizes an interface to the robot infrastructure for the individual services. The software architecture supports a modular design, where each subdialog (greeting, weather info, news, entertainment, etc.) is an independent module defining a service. Thus, it is easy to add new functionality and refer to, or combine existing dialog capabilities in new dialogs, such that the borders of the modules get blurry for the user, who is perceiving the robot as one personality. The software modules implement a

content specific back-end functionality as well as define the subdialogs needed. The configuration for a subdialog is mainly a definition of the state space S (variables holding user inputs and context information) and a set of output actions available $A = \{a_1, \ldots, a_K\}$ (see Fig. 2 upper part). For instance, actions are expressing a greeting multimodally or asking which website the user likes to see. In order to reduce the necessity for exploration in an unordered set of actions and in order to prevent from selection of irrational sequences frustrating the user, it is necessary to limit the set of selectable actions in each state S. In our system this is done by a manually designed decision tree over the state variables S. Each node in the tree decides between two alternative branches, each consisting of sets of possible actions or further subtrees. These sets allow to define options from which the dialog manager has to choose later on by means of the probabilistic planner. By defining trees with only one action in the branches, it is possible to realize deterministic dialog strategies as well. In that case, the system behaves like a Finite State Machine. For realizing action sequences, it is necessary to consider the history of executions of the actions in the dialog state. Therefore, each possible action a_k has a counter H_k, holding the number of executions of a_k since the latest activation of the subdialog.

The dialog state of an independent subdialog, besides the counters for the actions, comprises a set of variables $\{V_1, \ldots, V_N\}$ representing user inputs, but also system internal data and events that are of relevance for the decision on the next action of the robot. For example, the number of appointments to be reminded or the answer (yes/no) to the question if appointments should be listed are such state variables in a "reminder" subdialog. The variables have a specific range that can either be discrete or real-valued, which has to be considered later in the respective similarity functions. The range is defined by the type of a variable that also defines which inputs can be filled in it by the input interpreter. Additionally, all variables are labeled with a certainty value $\{C_1, \ldots, C_N\}$ that expresses to what degree the respective information is known, unconfident, or unknown. This, for example, allows to model the ambiguity of speech recognition inputs or other probabilistic observations.

Therefore, the state representation for one subdialog is a discrete vector:

$$S = (V_1, \ldots, V_N, C_1, \ldots, C_N, H_1, \ldots, H_K) \qquad (1)$$

A. Control Flow in the Dialog System

Having defined the structure of a subdialog, now the coordination of user inputs, multiple active subdialogs, and the system output generation are explained.

In general, a turn-based control flow is realized where user input turns and system turns alternate. Once a system output action is executed, the system waits until expected user inputs are recognized or until a timeout triggers a new system turn. All multi-modal user inputs are processed in parallel by the input interpreter and will update the dialog state of the respective subdialog. Inputs or internal events are filled in variables of all subdialogs that match the respective type. Special variables may activate a subdialog if they get filled by an input or event.

The dialog manager holds a stack of active subdialogs (see Fig. 2), where the top most is that one evaluated each time a system action is necessary (start of a system turn). The dialog manager evaluates the decision tree of the top most subdialog in the stack using its current state $S = s_0$ in order to get the possible action set for the current situation. If only one action is available, it is executed by the output renderer directly. The interesting part, where adaptation takes place, is given when multiple actions are allowed by the decision tree, and the probabilistic planning process, as described in section III-C, is triggered. This planning yields a probability distribution on the actions $P_{plan}(A)$, that maximizes the probability of reaching a system internal goal state while maximizing the user's rewards on the way.

Since the system does not know neither the user's rewards and the possible transitions in the dialog states nor the goal states that are defined by the actions in these states, there is a need for exploration additionally to the aim for exploitation of the knowledge allready acquired (exploration-exploitation dilemma). Furthermore, the progress in the phase model of the long-time interaction, introduced in the intro part of this paper, also has to be considered during action selection. Thus, two additional probability distributions are used for action selection. The first represents the number of executions of each available action $P_{count}(A)$ to enforce that all possible actions are tried out equally during exploration. The second distribution $P_{prio}(A)$ allows for consideration of a priority that is depending on the progress in the long-time interaction. In this way, in the beginning phase, more explanatory actions are selected, and in the stable phase only straight actions without additional help messages and proactive actions like offering services in a certain situation are recommended. Consequently, the three influence factors $P_{plan}(A)$, $P_{count}(A)$, and $P_{prio}(A)$ are combined, and the action to be eventually executed is selected by drawing from that resulting distribution.

When the action is executed, mainly screen and speech outputs are generated, that may refer to values of the variables in the subdialogs, and communicate content suitable for the current situation (asking questions, confirming inputs or giving answers). Also special actions, like activation of other subdialogs or canceling an active subdialog, are possible. If a new subdialog is activated, the former top most in the stack gets suspended and can execute one more action in order to react to that special situation. If the interrupting subdialog is finished, the suspended one returns to the top of the stack and gets resumed. This resuming is a special action that may be used to bring the user back into the context of the former conversation.

B. Modelling of Interactions

For the planning and adaptation, the system needs to represent knowledge on the history of interactions with the user. This is done by means of several probabilistic models which are described in the following. The dialog manager first builds up a persistent probabilistic transition model at runtime that is representing the probability of reaching a certain state S' given a predecessor state S and the executed action A. Here, user specific decisions and reactions are learned as well as the internal restrictions on the state sequences, such that the planning system does not need to be configured with that knowledge in before. The representation of that model is a weighted sum of samples $s_j = (S', S, A)_j$ each of them equipped with a weight w_j. These samples have a very high dimensionality (keep in mind the elements of the state vector S) and, therefore, it is very unlikely that exactly the same states

 ISBN: 978-1-61208-341-4

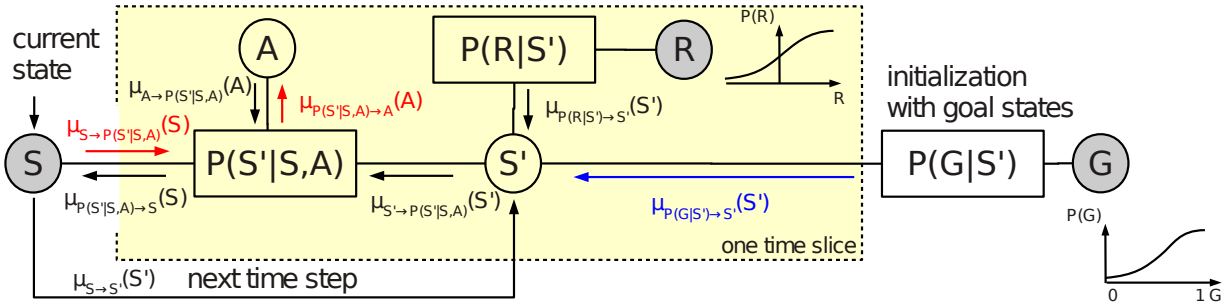


Figure 3. Dynamic factor graph for planning of next action A, given the current state S, a goal state distribution G and a reward proposal distribution R.

appear very often. To generalize consequences among similar states, a similarity function $\delta_{S',S,A}(s_a, s_b) \mapsto [0,1]$ defines a neighborhood among samples s_a and s_b. The subscript S', S, A determines the dimensions considered in that similarity. Later, when operating with marginal distribution,we need to compare samples on certain dimensions only. The probability of a particular transition $t = (S'_t, S_t, A_t)$, by means of that set of all samples s_j and similarity function is defined as follows with a normalization factor η:

$$P(S'_t, S_t, A_t) = \eta \sum_j w_j \delta_{S',S,A}(t, s_j) \quad (2)$$

For realizing a goal directed planning of action sequences, additionally a model of goal states $P(G, S)$ and a model of rewards $P(R, S)$ gained in a certain state, are defined similarly also as weighted set of discrete samples. Here, G can be 1 for a success state, e.g. the dialog was successfully completed, or it can be 0 for a fail state that has to be avoided in future interactions. This is for example the case, if a dialog turn times out without any user reaction.

The rewards are only recorded if positive or negative reward events took place. By ignoring zero rewards, the policy remains stable, even if the user is pleased with it and does not reward every action individually.

The goal and reward models can be used to evaluate a probability for a state S to be a goal state, as well as the probability to get a high reward in that state. The models return 0.5 if there are no similar observations in the sample set.

Initially, these user-specific models for each subdialog are empty and have to be filled during the interactions by observing and counting the real transitions, rewards, and occurrence of goal labels.

C. Probabilistic Action Planning

This section describes the planning mechanism used to deduce a probability distribution $P_{plan}(A)$ for the available actions, that maximizes the probability of reaching a goal labeled state while gaining as much reward as possible on the way to a goal state. For that kind of problems, the probabilistic model of our dialog is represented as a dynamic factor graph (see Fig. 3) for which a message passing algorithm called max product algorithm [14][15][16] exists. The max product algorithm can find the marginal distributions for all unobserved variables in the factor graph that maximize the probability given the set of fixed or observed variables.

The idea of that algorithm is to perform local operations in the nodes of the factor graph and propagate the results in form of messages $\mu_{sender \to receiver}(domain)$ along the tree structure of the graph in order to get the result in the node of interest, which is the A node of the first time step in our case. Unfortunately, in our dynamic factor graph, the number of time steps to reach a goal state is not known in before, but, since we are only interested in the next action, the inference can be executed in a loop, with one iteration for each time step to be looked in the future. This is shown by the message $\mu_{S \to S'}(S')$ in the figure. This is only possible in an acyclic factor graph, which is the reason for the complex state S and the respective factor models we have chosen in the model. A further factorization would possibly simplify the factor potentials, but the complete factor graph would not longer be acyclic afterwards. This would make the time step loop trick impossible since belief propagation needs to iterate on the complete structure of a loopy factor graph.

A central decision for the factor graph algorithm is the form of representing the probability distributions in the nodes of the factor graphs and in the messages sent between them. Normally, Gaussians or discrete distributions are used for that, but in our case the dimensionality of the distributions' domains is too large for that. Hence, we propose a representation as mixture of discrete samples as already introduced for building up our transition model $P(S', S, A)$. In the following, we briefly show, how the operations required by the max product algorithm have been implemented for that kind of distribution representation.

The **initialization** of the planning loop requires setting the distribution of the state S' to the desired distribution of goal states we expect. For that, the model of goal probabilities $P(G|S')$ for each observed state provides the fraction of occurred goal labels in the actions leading to that state. Asuming, that the target probability for reaching a goal is distributed like the little diagram shown in Fig. 3 bottom right, the $P(G|S')$ table is used to weight the states S' of the transition model, thus a set of weighted S' samples results. This is our initial state distribution $P(S')$ of the last time step in the planning horizon (blue message $\mu_{P(G|S') \to S'}(S')$).

Then the planning loop starts with the **step 1**: According to the max product algorithm, the distribution $P(S')$ has to be multiplied by the message $\mu_{P(R|S') \to S'}(S')$ from the reward model. This reward model $P(R|S')$ is also a set of state samples s_i, that maps to the average reward r_i gained in that state. We use the proposition, which is maximizing the reward along the sequence of states, and define the distribution of the expected reward as a sigmoid function (shown in the little diagram besides the R node in Fig. 3). Knowing this, for each sample s_j in the S' distribution, we can compute an

updated weight $w_j^{(S')}$ by comparing the states of these samples to states of the samples s_i in the reward model and applying the sigmoidal proposition distribution indicating the desirability of the respective reward. Adding an ϵ ensures that weights will be vaild, even if no samples of the reward model do match in the similarity function.

$$w_j^{(S')} = w_j \frac{0.5\epsilon + \sum_i \frac{1}{1+e^{-r_i}} \delta_{S'}(s_j, s_i)}{\epsilon + \sum_i \delta_{S'}(s_j, s_i)} \quad (3)$$

To reduce the complexity for the next computations, samples with low weights are omitted, and samples with high similarity $\delta_{S'}(s_j, s_i)$ are merged until a maximum number of samples remains. We used 20 samples at maximum in all message distributions.

In **step 2**, this sample set $P(S')$ is sent to the transition model as message $\mu_{S' \to P(S'|S,A)}(S')$ to get multiplied to the conditional probability distribution $P(S'|S, A)$.

Since we only have counters $w_j^{(S',S,A)}$ for the occurrences of the various state transitions, we need to divide $P(S', S, A)$ by $P(S, A)$ to get the conditional, which is done by calculating new weights $w_j^{(S'|S,A)}$ for the samples s_j according:

$$w_j^{(S'|S,A)} = \frac{w_j^{(S',S,A)}}{\sum_i w_i^{(S',S,A)} \delta_{S,A}(s_j, s_i)} \quad (4)$$

This can be done before planning starts and has only to be updated when a new transition has been observed.

In the $P(S'|S, A)$ node, the product of the factor potential and all incoming messages has to be calculated. Therefore, each sample of the transition model gets a new weight $\hat{w}_j^{(S'|S,A)}$ (5), which incorporates the similarities of samples s_i in the incoming message $\mu_{S' \to P(S'|S,A)}(S')$. The message $\mu_{A \to P(S'|S,A)}(A)$ is assumed to be uniform and is not considered in the weight computation. It would be possible to incorporate priors on actions here to realize a dependency of the actions on the progress in the long-term interaction phase.

$$\hat{w}_j^{(S'|S,A)} = w_j^{(S'|S,A)} \sum_{i \in \mu_{S' \to P(S'|S,A)}(S')} \delta_{S'}(s_j, s_i) w_i^{(S')} \quad (5)$$

The max product algorithm now needs to find the maximum probability for each value of the variable for the outgoing message. In our case, that is the $\mu_{P(S'|S,A) \to S}(S)$, and thus the goal variable is S. For each sample, all other samples are compared using $\delta_S(s_i, s_j)$, and only the one with the maximum weight will be used. As result, we get a message $\mu_{P(S'|S,A) \to S}(S)$, that can be processed further in the variable node S.

Here, in **step 3** we have to test, if the current state s_0 of our subdialog matches the inferred predecessor state distribution $P(S)$ for the planned sequence to the goal state of a length of the current iteration. In case of sufficient probability, the action A can be deduced, that maximizes probability of going one step towards the goal state starting from s_0. This is done by sending the message $\mu_{S \to P(S'|S,A)}(S)$, which is simply our current state s_0 with weight $w_0 = 1$ to the factor node (red message in Fig. 3).

There, again the product and maximization has to take place, which is realized by re-weighting the samples of the transition model again. Here, the intermediate weights already containing the message from S' can be reused.

$$\tilde{w}_j^{(S'|S,A)} = \hat{w}_j^{(S'|S,A)} \delta_S(s_j, s_0) w_0 \quad (6)$$

With these new weights, the maximization along the S and S' dimensions can be done in oder to get a probability for each action to reach a goal within the current time horizon. This is done by grouping the samples of the transition model s_j by the discrete action dimension A and checking for the maximum weight $\tilde{w}_j$ in each group.

Before going to the next planning time step by sending the $\mu_{S \to P(S'|S,A)}(S)$ message renamed as $\mu_{S \to S}(S)$ and starting over with step 1, for each action in the set of available actions, the maximum probability over all time steps is stored. Afterwards, this is used to gain the $P_{plan}(A)$ distribution used for action selection as described before.

D. Prediction of User Preferences

In many situations in a repeatedly conducted dialog between the robot and the user, the annoying questions for options (e.g., which website should be shown) can be omitted, when using the former choices in a similar situative context. The transition model we built from the dialog history, exactly contains these information.

Thus, instead of asking the user for the information, the robot can try to infere the desired value, which is changing the state S of the subdialog similarily without any further inputs from the user. Depending on the outcome of that, the dialog can continue either with a confirmation of that fact or with a question for specification of the information, if the inference did not yield a significant probability for either option.

By means of that, also the proactive situation dependent offer of services can be realized easily, where the correctness of the suggestions strongly depends on the context variables considered in the subdialogs state vector.

IV. Experimental Evaluation

The correct function and ability of considering observed reactions of the user during robot's action selection first has been validated by means of a set of simulated dialog sequences not embedded in a complete robot application. This functional test showed the learning capabilities as expected, but it is hardly possible to simulate a more complex application realistically. Even more, it is not possible to predict the impact of our dialog system on real users.

Due to these circumstances, the proposed system has afterwards been evaluated in a separate application before being applied for realizing the user interface of our health assistant robot. Caused by restrictions on access to the robot as well as the number of test users needed in combination with the intended long-term interaction, an evaluation scenario has been chosen, that involved 16 members of our lab in parallel. The robot has acted as an "Office Mate" by visiting each trial participant once a day and offering its services after the user confirmed his/her supposed identity and the respective persistent interaction models have been loaded. The experiment took ten workdays in order to give time for an observable change of the robot's behavior.

The participants had to fill out two questionnaires with a first focus on the long-term acceptance of a robot with an adaptive dialog behavior, and a second focus on the practicability of

 ISBN: 978-1-61208-341-4

our realization. Questions mainly followed the Almere Model [17] , while questions regarding the perception of the robots adaptation skills have been added.

The first questionaire had to be filled in before the experiments to get the baseline and find preferences regarding the form of being addressed by the robot (formal or informal), as well as to get hints on the set of websites to be provided by the robot during the interactions. A second questionnaire after the experiment asked for the personal experience of the participants.

The tests have additionally been accompanied by an observer to reveal usability problems, and aforementioned questionaires have been used for a qualitative evaluation due to the low number of participants.

A. *Office Mate Services*

The application for our Office Mate scenario covers a couple of services. For giving an impression on the capabilities and the options for adaptivity, these services are described in the following.

The adaptivity was mainly realized in the **main menu** subdialog, which had a set of optional actions that comprise (M1) offering a normal selection menu, which of the services to execute next, (M2) proposing an unused service to introduce it to the user with the question whether to start it or not, (M3) executing a service proactively depending on the prediction of a selection. The user in that case only has to confirm or deny, whether that the service is going to be started. The context state variables for the prediction of the next service to be selected by the user include one counter for each service available counting the number of activations of that service in that session. By means of these context variables, the system can learn arbitrary sequences of services used.

However, actions (M4) to (M6) are the same as (M1) to (M3), except that a written advice on how to interact with the robot in that situation is given on the touch screen additionally. These later actions are designed for the first phase in the long-time interaction, where the user should learn how to use the robot.

Further variability is implemented in the **greeting** subdialog, which is always started at the beginning of an interaction session with a participant. Besides a deterministic greeting, here the optional actions were: (G1) giving a tutorial on how to use the robot, (G2) asking for wellbeing, and (G3) quitting the greeting subdialog and continuing with the main menu subdialog. During the greeting subdialog, a sequence of these actions is also possible.

The first two services offered are **news** and **entertainment** via websites in a browser. These subdialogs had optional actions as well, which were (N1) presenting a longer list of respective websites for selection, or (N2) suggesting a website based on the predicted selection known from previous interactions. Since the number of different websites visited is part of the state variables of that subdialog, it is possible that the system can learn and predict a sequence of sites preferred by the user. Also in these services, the available actions could be executed with an additional advice on how to use the screen menu.

The third service, a **weather forecast**, had the only option to present weather warnings, if available, automatically or to wait until being asked for. In all cases, the current temperature and weather conditions as well as a two day forecast are presented on the screen.

Two more rather simple services are answering questions for current **date and time** as well as showing the **menu of the refectory**. Since these services do not require further decisions by the user, there are no optional actions.

A last service offered was a **reminder service**. The user was able to edit and show appointments in a list for the current week or the current day. On the main menu, there was an indication on the number of reminders for that day. Since this number is also used in the context variables of the main menu, the proactive presentation of reminders was possible if the user teached that behavior by selecting the reminder presentation manually some times. Unfortunately, the calendar was not synchronized with the Google Calendar usually used in our lab. Therefore, that service had not been used consequently by the participants.

To quit a session with the robot and send it to the next user, the option for a good bye dialog was available in the main menu.

B. *Results and Discussion*

When asked for their expectation to the Office Mate robot, among a couple of additional features for such a robot (sending it to others, sending it to get coffee, using it as avatar by a video conference), few generic aspects regarding adaptation have been mentioned by the participants beforehand. One aspect was that the robot is supposed to know when the user can be disturbed and when not. Unfortunately, that is only possible if detailed context information on the user and the situation at all is available to the robot.

Not all participants of the experiment have been available all the ten days; thus a difference in the experience for different durations of interaction could be observed. All participants who had more than four interactions did notice that the robot learned their preferences and also changed its behavior over time. That mostly is related to the prediction of user's choices (see III-D), leading to a suggestion of following services if the user's attitude is stable. Here, the robot developed an individual sequence of suggestion of services from the main menu as well as individual sets of websites preferred for each user.

Unfortunately, most of the users mentioned, that they were a bit confused by the option of rewarding the robot. They wanted to reward explicitly special aspects of the complex behavior like diction or level of advice. It was not transparent to them, what aspects are variable in this situation and to which aspect the reward refers. This is an indication for using more implicit and system-internally generated rewards in future, that the users are not aware of. By means of that, the user does not need to know the alternatives the robot has in certain situations. Concluding this, offering explicit good/bad buttons is not a practical way for getting rewards from the user.

Those users who used the reward buttons could also influence the dialog flow and the way of presentation of information. If the tutorial in the greeting subdialog had been punished, it occured less often during the following days. Also, the written advice on how to interact in certain situations appeared less often during the experiment if the user punished that behavior by means of negative rewards. However, the user's perception of the exploration of alternative actions is critical. It was confusing for many users, that the robot, although they had already rewarded a behavior, acted again

 ISBN: 978-1-61208-341-4

in that unwanted way. The reasons for this were the complex state model on the one hand and the exploration strategy on the other hand. In many situations, the user might think that s/he already had passed exactly that sequence, but in the robots state variables there is a difference in the history or in the certainty values, causing that the situation is unknown to the robot, thus an exploarative action is selected that the user confused. Additionally, even if the robot knows the situation from past interactions, the need for exploration of longer sequences of actions yields further executions of unwanted actions regardless of the history in the reward model.

For a better generalization over states that are different in only a few dimensions, the planning step in the transition model needs to be improved in order to recombine parts of samples. That will help to reduce the number of observations the system needs to learn a possible path through the state space. Furthermore, it may be possible to apply a more greedy action selection strategy as proposed in the TAMER system [8] to reduce unexpected repetitions of actions that result from exploration steps. Unfortunately, this conflicts with the ability to find an action sequence that reaches system internal goals.

An additional possibility for reducing the number of rewarding events and improving generalization skills is the handling of recurring variations of actions in different situations. In the Office Mate implementation, there was an alternative with or without additional output in many situations. It would be better to introduce a global property "use additional support outputs" which is followed if possible and switch it on and off by only one action instead of generating almost similar copies of arbitrary actions. Therefore, the negative reward for one output in situation A could influence the form of output in situation B without having seen situation B in before. Besides the degree of advice needed, also the volume of voice outputs or the form of addressing the user (formal or informal) are candidates for such global properties.

The overall results showed that the planning algorithm does work very well, but the way of configuring the subdialogs has to be improved in future real applications considering the findings discussed above. More effort has to be put in the deduction of meaningful reward from the user's reactions on robot's behavior. Also the perception skills of the robot always could be improved to extend the context variables of the subdialogs. By means of that, proactive behavior of the robot can be better suited to the actual situation.

V. Conclusion

We could show in an experimental setup, that a dialog manager using an online life-long learning transition model for online planning of action sequences can be realized. The very high dimensional state space of a human robot dialog can be managed by splitting the whole application into independent subdialogs. This also improves the application development process by modularization. An efficient way for the probabilistic inference during the planning process could be realized by a sample-based representation of probability distributions. At last, independent of the planning algorithm, we could identify a couple of design issues with our test application, that have to be improved in future implementation.

Acknowledgment

This work has received funding from the Federal State of Thuringia and the European Social Fund (OP 2007-2013) under grant agreement N501/2009 to the project SERROGA (project number 2011FGR0107).

References

[1] "SERROGA webpage", 2014, URL: http://www.serroga.de [accessed: 2014-04-01].

[2] H.-M. Gross, C. Schröter, S. Müller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner, and M. Merten, "Progress in Developing a Socially Assistive Mobile Home Robot Companion for the Elderly with Mild Cognitive Impairment", in Proc. IEEE/RJS Int. Conf. on Intelligent Robots and Systems (IROS 2011), pp. 2430-2437.

[3] H.-M. Gross, C. Schröter, S. Müller, M. Volkhardt, E. Einhorn, A. Bley, T. Langner, M. Merten, C. Huijnen, H. van den Heuvel, and A. van Berlo,"Further Progress towards a Home Robot Companion for People with Mild Cognitive Impairment", in Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (IEEE-SMC 2012),2012, pp. 637-644.

[4] T. Blaise, J. Schatzmann, and S. Young. "Bayesian update of dialogue state for robust dialogue systems." In Proc. of Acoustics, Speech and Signal Processing, ICASSP 2008, 2008, pp. 4937-4940.

[5] M. Toussaint, S. Harmeling, and A. Storkey, "Probabilistic inference for solving (PO)MDPs", Informatics Research Report 934, School of Informatics, University of Edinburgh, 2006.

[6] J. Pineau and S. Thrun. "High-level robot behavior control using POMDPs." AAAI-02 Workshop on Cognitive Robotics, Vol. 107, 2002.

[7] W. B. Knox and P. Stone. "Interactively shaping agents via human reinforcement: The TAMER framework." In Proc. of the fifth international conference on Knowledge capture. ACM, 2009, pp. 9-16.

[8] W. B. Knox, P. Stone, and C. Breazeal, "Training a Robot via Human Feedback: A Case Study." In: Social Robotics. Springer International Publishing, 2013, pp. 460-470.

[9] H. M. Meng, C. Wai, and R. Pieracciniet, "The Use of Belief Networks for Mixed-Initiative Dialog Modeling", In Transactions on Speech and Audio Processing, 2003, vol. 11, no. 6, pp. 757-773.

[10] F. F. Martinez, J. Blzquez, J. Ferreiros, R. Barra, J. Macias-Guarasa, and J. M. Lucas-Cuesta, "Evaluation of a spoken dialogue system for controlling a Hifi audio system", In Proc. of Spoken Language Technology Workshop, IEEE SLT 2008, 2008, pp. 137-140.

[11] R. Parr, and S. Russell, "Reinforcement learning with hierarchies of machines." in Advances in neural information processing systems, 1998, pp. 1043-1049.

[12] "MIRA website" URL: http://www.mira-project.org [accessed: 2014-04-01].

[13] E. Einhorn, R. Stricker, H.-M. Gross, T. Langner, C. Martin,"MIRA - Middleware for Robotic Applications" in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2012), 2012, pp. 2591-2598

[14] C. M. Bishop, "Pattern Recognition and Machine Learning", New York: springer, 2006.

[15] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm", In: Information Theory, IEEE Transactions on, 2001, vol. 47, no. 2, pp. 498-519.

[16] H. A. Loeliger, "An introduction to factor graphs", in Proc. Signal Processing Magazine, IEEE, 2004, vol. 21, no. 1, pp. 28-41.

[17] M. Heerink, B. Kröse, V. Evers and B. Wielinga, "Assessing Acceptance of Assistive Social Agent Technology by Older Adults: the Almere Model", International Journal of Social Robotics, 2010, vol. 2., no. 4, pp. 361-375.

 ISBN: 978-1-61208-341-4

A Black Box Validation Strategy for Self-adaptive Systems

Georg Püschel, Christian Piechnick, Sebastian Götz, Christoph Seidl, Sebastian Richly, Uwe Aßmann
Software Technology Group, Technische Universität Dresden
Nöthnitzer Str. 46, 01062 Dresden, Germany
Email: {georg.pueschel, christian.piechnick, sebastian.goetz1, christoph.seidl, sebastian.richly, uwe.assmann}@tu-dresden.de

***Abstract*—Self-adaptive systems are able to operate autonomously by reconfiguring themselves for changing context conditions and tasks. This capability requires a process of decision making that can only be partially hard-coded. Some parts of the logic are the result of reasoning and, thus, implicit to the system designer or user. In consequence, the quality of the systems functionality has to be extensively validated before delivery. During the validation, firstly, the response of adaptation decisions as a result of environment change has to be examined. Secondly, it is necessary to check the interaction of adaptation and non-adaptation-related behavior. The management of all this information is expensive. Therefore, we propose an approach that separates environment change, functionality and adaptation concerns using expressive models. The models are executed by a simulator and validated against the real behavior of the system under test. We illustrate the complete approach using an example SAS operating a domestic service robot. Our design process and the proposed modeling principles equip engineers with a toolset that allows them to face the challenging complexity of self-adaptive system validation.**

Keywords*—*self-adaptive systems; service robots; model-based testing; simulation; feedback loops

I. INTRODUCTION

A Self-adaptive System (SAS) [1] adapts itself according to changes in its environment. The continuous execution of sensor monitoring, decision making, planning, and adaptation execution is organized in feedback loops [2]. Due to the use of intelligent reasoning strategies, the SAS is capable of fulfilling its tasks more efficiently or it even may find solutions to tasks that were not explicitly defined at design time.

In our work, we aim to provide solid SAS development methods and, thus, we also require a validation approach that is able to deal with the complexity of such self-adaptive behavior. The mechanisms that decide autonomously have to be validated extensively before deploying the system in a productive environment. A limitation is that a SAS can be adapted from external or reason about unanticipated events can never be tested comprehensively in this phase of the life cycle.

However, even for these systems, the user's trust has to be gained by examining the system in an appropriate variety of scenarios. Hence, validation methods can be performed on different abstraction layers as, for instance, the German V-Modell [3] proposes. On the lowest abstraction layer of modules, knowledge of code and design models can be utilized. However, due to the complexity and large variety of possible situations, performing a comprehensive validation (e.g., by deriving and executing test cases) on these levels is expensive.

In contrast, validating SAS applications on acceptance level, based on requirements of a more abstract specification, is more promising. For this purpose, the engineer no longer relies on detailed knowledge of the system interior but on a black box interface that is used to enforce situations and validate the outcome. Thus, setting up a black box interface that provides all necessary operations to interact with the system and to query information that has to be examined, is the first crucial task during the validation phase.

A validation method for specification-based black boxes is model-based testing [4]. In this approach, a test model is specified and test cases are generated from it. Additionally, a further problem is that SAS can be deployed in complex environments where not every detailed situation can be enforced. For instance, some entities the system is interacting with like hardware controllers or physical objects are difficult to be formalized. Instead, the test model designer may specify some future decisions depending on run-time state that is observed from these entities at runtime. As sequential test cases cannot support such decisions, the model has to be executed at run-time. Therefore, we propose using simulation and capturing the discussed non-specifiable parts of the system or test environment *"in-the-loop"*.

The challenge in simulating a SAS is to provide a meta-model that is expressive enough for compactly specifying all behavioral and adaptation-related aspects. These aspects are given by several requirements that we derived from failure scenarios in our previous work [5]:

(1) Correct sensor interpretation
(2) Correct adaptation initiation
(3) Correct adaptation planning
(4) Consistent adaptation/system interaction
(5) Consistent adaptation execution
(6) Correct system behavior (especially actuator actions)

Goals (1) and (6) include the validation of the correctness in sensor perception and actuator control. Both properties can be checked in isolation by instrumenting the respective drivers. However, in this paper, we focus on the goals (2)-(5), which directly deal with the SAS feedback loop (sometimes referred to as MAPE loop: monitor, analyze, plan, execute [2]). In order to match the requirements, the model has to provide means for defining in which situations an adaptation has to be initiated (goal 2), how the system has to adapt (goal 3), how the adaptation has to be scheduled with non-adaptation-related behavior (goal 4), and how the end result of the adaptation is expected to look like (goal 5).

In order to match these requirements, we contribute a methodology to separate their different aspects in a composite simulation model. Parts of our model are enriched with *assertions* on the System Under Test's (SUT) interface in order to define how a simulation state has to be concretely validated. We illustrate the complete modeling methodology using our HomeTurtle domestic robot. In the HomeTurtle scenario, a robot is deployed in a flat of a handicapped person and is

 ISBN: 978-1-61208-341-4

capable of delivering various items, which are stored in a software-controlled cabinet.

The remainder of this paper is structured as follows: In Section II, we start with our example adaptive system. In Section III, we present our approach based on this example. In Section IV, we illustrate an example simulation run. In Section V, we present out implementation and experimental environment. Afterwards, in Section VI, we discuss related work. In Section VII, conclusion and future work are discussed.

II. Example Application: Home Turtle

In this section, we present an illustrative example of a SAS that supports a handicapped person at home. The scenario is depicted in Figure 1. A service robot "HomeTurtle" (an extended version of the TurtleBot platform [6]) is initially deployed in the flat. The task of the robot is to find and deliver a desired item to the user (i.e., the inhabitant). Those items can be dropped from a *cabinet* into a basket mounted on top of the robot. Therefore, the cabinet contains several boxes with magnetically clamped flaps. The magnets are triggered from a WiFi-connected embedded device.

In the beginning, a user instructs the robot by entering the desired item (e.g., "towel") using a Tablet PC that is accessible nearby. Using a wireless network, the robot can query the flat's map, available cabinets including their positions and contents. After this information has been gathered, the robot is able to inform the user whether the desired item is available. If the item has been found, a route is planned and the robot starts driving. In this process, the robot has to avoid collisions with walls and other obstacles (symbolized by office chairs). After approaching a cabinet and parking in a predefined position underneath it, the robot signals the cabinet to drop the requested item. Afterwards, it drives back to the user. Additionally, during the complete process, the environment may signal an emergency (e.g., a fire or medical emergency). In this situation, the robot is expected to drive to its emergency position as labeled in our illustration. Thus, it avoids to obstruct the access of human helpers to the inhabitant.

The following sensors and actuators are used to accomplish the robot's task:

- **Robot drive:** The robot drive has three modes for stopping (0=stop) and driving in arbitrary directions with two different velocities (1=slow, 2=fast).
- **Stereo camera:** Can be used to recognize walls, obstacles, and the cabinets.
- **On-board computation unit:** The robot runs its operations on-board using a fix-installed netbook that connects to all the hardware on the robot.
- **Smart illumination system:** The flat is equipped with room lights that can be operated by the software system to improve the flat's illumination.
- **Local WiFi:** The robot, as well as the cabinet, are connected to a wireless network. Thus, the flat's map and information about the cabinet's position and contents can be shared.

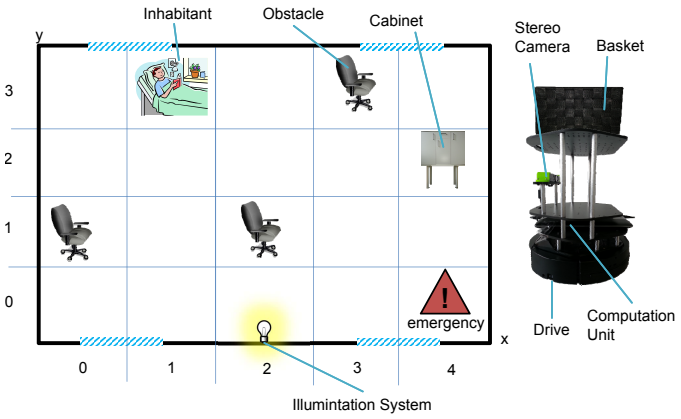


Fig. 1. Scenario: HomeTurtle operating in a flat.

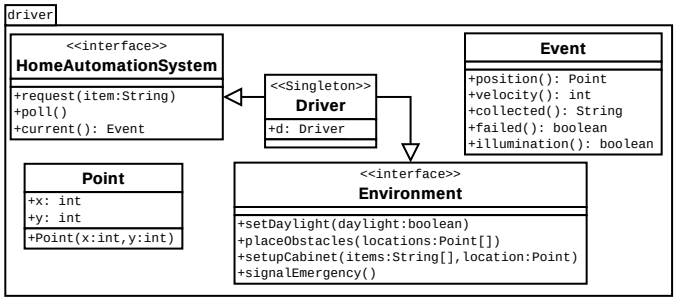


Fig. 2. Test driver interface.

Furthermore, to improve its behavior, assure safety and minimize operation time, the following *adaptations* are possible:

- **Improve illumination:** If the robot enters a room and daylight from the windows is not sufficient for object recognition, the robot connects to the illumination system and activates it.
- **Location-dependent velocity:** While driving at fast mode velocity, the robot is not able to stop in time if an obstacle is detected. As the obstacles' positions may change, the robot is expected to run in slow mode during the current request as long as the current position was not explored during this request.

In order to send input data to the real system and to validate its output, the simulation has to communicate with the system using a test driver. For our example, we implemented such a driver whose interface is depicted in Figure 2. The `Driver` holds a static instance `Driver.d` and implements two interfaces: Firstly, `Environment` provides methods to enforce an emergency signal, mock a light state, and setup obstacles and a cabinet. In order to reduce the scenario's complexity, we assume that the positions of the inhabitant and emergency locations as well as the room's layout are static. Secondly, the interface `HomeAutomationSystem` can be used to request a new item from the robot or to retrieve events that can be validated during simulation. Each `Event` captures information about the current position, velocity, and illumination. It also informs whether an item was collected or the search has failed.

 ISBN: 978-1-61208-341-4

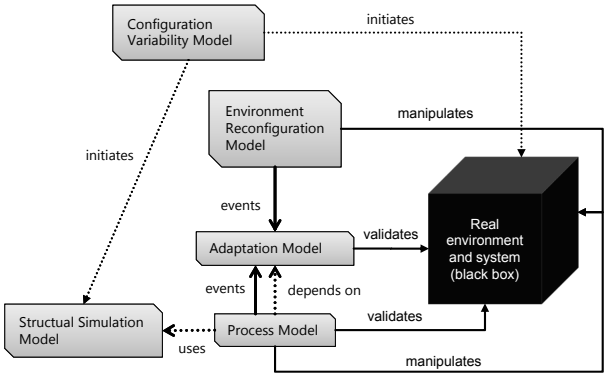


Fig. 3. Concern-separated components of the simulation model.

III. Validating SAS by Using an Adaptive Simulation Model

In this section, we present our methodology. The briefly discussed challenges are tackled in different components of a black box simulation model. These components, as well as their dependencies, are depicted in Figure 3. Each component matches a set of specific concerns that were separated in order to decouple the responsibilities during the design process. The model is as much as possible based on Unified Modeling Language (UML) 2 [7], Object Constraint Language (OCL) [8] and a special version of equivalence class trees [9].

The recent state of the performed scenario is reflected by the *Process Model* that is based on state charts. The actions performed during execution are, firstly, the requests that are sent to the test driver and, secondly, *assertions* that determine whether the received events are correct in the current state. Thus, the state of the simulation model represents assumptions on the state of the real system. In order to work with more detailed state-defining information, the *Structural Simulation Model* (i.e., a UML class model) is used. During the initiation of the system, the environment is set up and, synchronously, the Structural Simulation Model is configured with information that reflects this initial environment setting. As there may be different variants of initial configurations, the *Environment Variability Models* defines an equivalence class tree that allows to derive such configurations. The *Environment Reconfiguration Model* contains state charts with actions that define environment manipulations in order to trigger adaptation in the real system. As it defines an operational order of manipulations, requirement (3)–*correct adaptation planning*—can be dealt with. Regarding the requirement (2) (cf. Section I), it has to be validated whether system correctly adapts to these changes. Therefore, the Environment Reconfiguration Model produces events that are consumed by an *Adaptation Model* that reflects adaptation modes and validates them using assertions (requirement (5)–*consistent adaptation execution*). This Adaptation Model is a state charts as well. Events can also be produced by the Process Model and its behavior can be tailored to the Adaptation Model's state. Thus, requirement goal (4)–*consistent adaptation/system interaction*–is matched. The details of the individual model components are explained in the following.

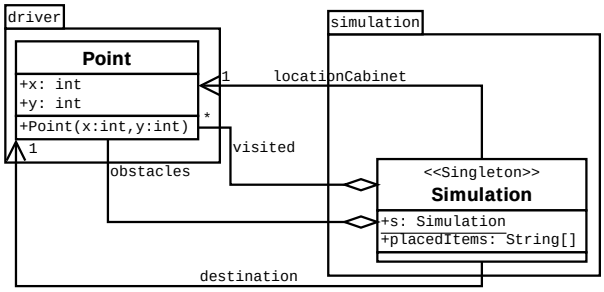


Fig. 4. Structural simulation model.

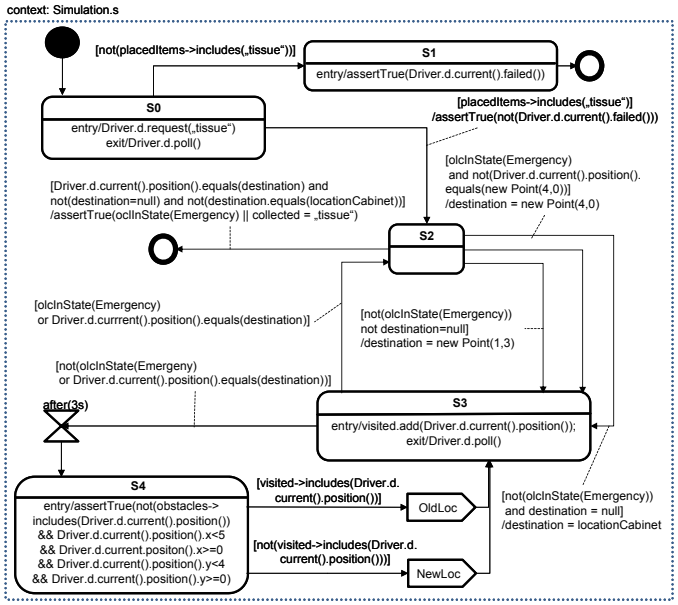


Fig. 5. System process model.

A. Structural Simulation Model

During the simulation, several assumptions on the real system have to be managed that are represented by a simulation state. For our example application, the locations of obstacles and the cabinet has be remembered as well as the locations that were already visited. This state is captured by a structural model as depicted in Figure 4. The singleton object `SimulationState.s` holds attributes and aggregates objects that can be manipulated or evaluated by the central System Process Model.

B. System Process Model

The *System Process Model* defines the task-specific behavior of the system and how it interacts with its adaptation feedback loops. For our example, we defined these aspects in an UML State Chart as depicted in Figure 5. It uses OCL constraints whose context is the static instance `Simulation.s`. In state `S0`, a request for a towel is initiated and the first event is polled. If the initial configuration set up the cabinet with the desired item, `S1` is reached, otherwise `S2`. The action of the latter transition (i.e., the entry action of `S1`) performs an assertion on whether the real system has either failed or not. If any assertion in the models fails, the simulation is cancelled and an

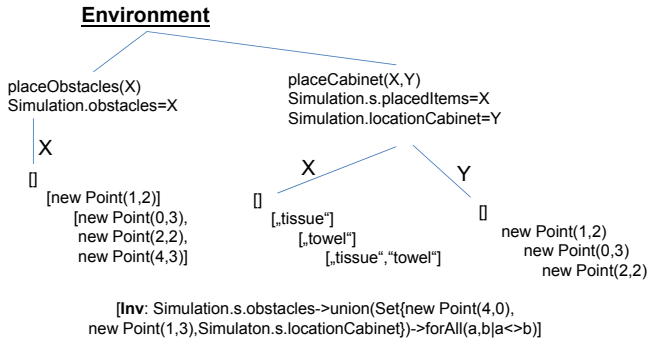


Fig. 6. Environment configuration variability model.

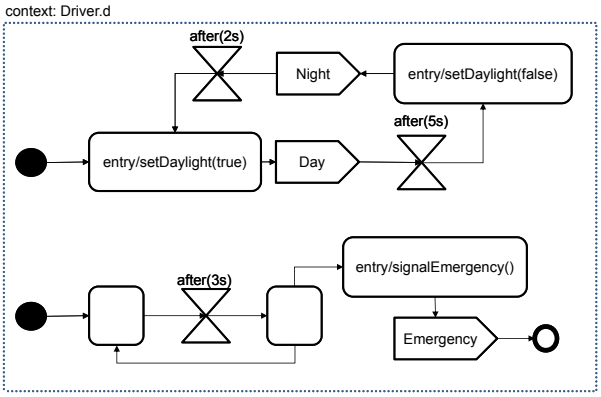


Fig. 7. Environment reconfiguration model.

error is signaled. Starting from state `S2`, the robot's destination is determined by evaluating the previous destination value (either null, the start place, the cabinet's place or the emergency position).

States `S3` and `S4` form a feedback loop. When entering `S3`, the current position is appended to the list of visited locations and the next event is polled. In the next step, the loop sleeps three seconds (indicated by the `AcceptTimeAction`, cf. UML spec. [7]). Thus, the Adaptation Models are expected to enforce changes to the environment that are interleaved with the process. Subsequently, in `S4` an assertion is performed in order to ensure no obstacle has been hit and the robot did not leave the boundaries of the scenario. Depending on whether the current position is contained in the `visited` collection, a signal `OldLoc` or `NewLoc` is produced. Therefore, we use the `SendSignalAction` UML element. These signal events are later used to synchronize with the adaptation models. At this point, the feedback loop is restarted. As soon as the destination is reached, the transition to state `S2` is triggered. Another exit possibility from the loop is triggered when the `Emergency` adaptation mode is active. This information can be queried by the `oclInState(...)` function, which is applied to the Adaptation Models. In this way, an interaction between the task-related process and the adaptation mode of the SAS can be modeled. The final state is enabled if the robot reaches a destination that is not the location of the cabinet. The respective transition checks an assertion whether either an emergency was signaled or the correct item was collected.

C. Environment Configuration Variability Model

The state space of an environment situation can be enormously large. In testing, this problem is usually dealt by using classification. For instance, data ranges of the system's input parameters are split into equivalence classes and only representatives are tested. All representatives of an equivalence class are assumed to produce the same output. For our example, we designed a special model as depicted in Figure 6. The hierarchical structure serves as a decision tree for determining under which initial conditions a simulation can be started. Each one of the `Environment` child nodes performs multiple operations: Firstly, the real system is initiated (e.g., the robot is set up in its initial location) and secondly, the simulation state is manipulated such that it reflects this initial configuration. The operations are parameterized with one or two substitution variables. Each variable can be replaced by one of the concrete values in its leaf nodes. The latter ones are the equivalence class representatives. Furthermore, the model contains an invariant to prohibit configurations where the robot's start position, obstacles, or the cabinet are put in the same location.

Basically, this model represents the variability of possible environment settings. Thus, more sophisticated models of variability (e.g., attributed feature models [10]) can also be used for the same purpose. Inherent invariants of such models can restrict the configuration variability space to a manageable size. However, a specific challenge of SAS is to validate whether to system adapts correctly the changes of this configuration. Therefore, in the next section, the configurations dynamics are defined.

D. Environment Reconfiguration Models

Figure 7 depicts a simple model of environment reconfiguration. In the upper chart, the entry point of the first state sets the environment daylight to `true`. The driver is now in charge of mocking the brightness sensor's input data and thus enforces the system to adapt. In order to reflect the expected adaptation in the simulation model, a signal `Day` is produced that later will be received by the Adaptation Model. After five seconds, the daylight setting is inverted and the `Night` signal is sent. After additional two seconds, the reconfiguration loop restarts. The lower chart performs a loop that every three seconds demands the simulation to decide of an emergency is signalled or not. This decision can, for instance, be determined randomly or by the user.

Using such environment reconfiguration models, scenarios with different operational orders can be generated. Based on these scenarios, the SUT is stressed and its reactions are exhaustively validated. Using timing, the variety of interleaving possibilities with actions from the Process Model can be reduced.

E. Adaptation Model

Adaptation models define how a configuration has to be altered in response to a received signal. Signals have been produced by either the Environment Reconfiguration Models or by the Process Model in order to notify about a condition that may cause an adaptation. Figure 8 depicts three state charts for the velocity, illumination, and emergency adaptations.

 ISBN: 978-1-61208-341-4

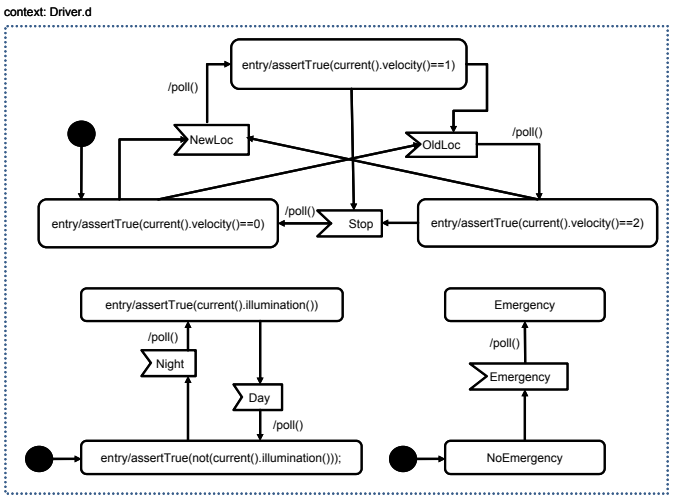


Fig. 8. Adaptation models.

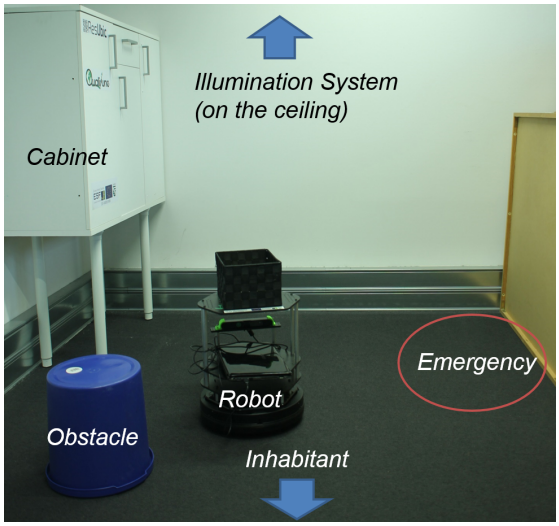


Fig. 10. The HomeTurtle lab.

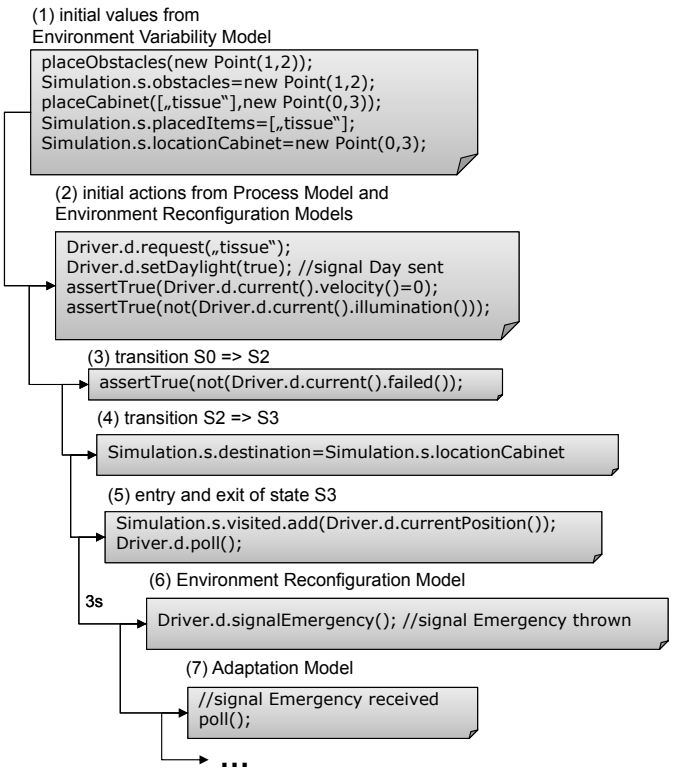


Fig. 9. Excerpt of an example simulation run.

States of an adaptation state chart may contain an entry operation, which performs a validation on the system's adaptation mode. Using UML `AcceptEventActions`, the automaton is designed to wait for the signals. After a signal was received, a new system event is retrieved (`poll()`) such that the assertion is performed on a fresh information basis. Each Adaptation Model stores a specific aspect of the SUT's adaptation mode. Behavioral adaptations are defined using constraints on the Adaptation Models' states.

IV. Simulation

To clarify the models' interactions, we illustrate an excerpt of an example simulation run in Figure 9. The simulation is indeterministic as there can be several execution paths. Sequence (1) of operations is generated by the Environment Variability Model. The simulator automatically selects a solution of the model's invariant such that no obstacle position equals the positions of the inhabitant, cabinet, or emergency stop. When the different state charts are initiated, operations sequence (2) is performed as defined in the initial states. When the Environment Reconfiguration Model sets the daylight property, a signal `Day` is produced. However, as the respective Adaptation Model has no matching outgoing transitions in its initial state, this signal is ignored in this specific state. Sequences (3) and (4) are generated when the transitions `S0->S2` and `S2->S3` are triggered. `S0->S1` cannot be executed as tissue item was placed in the cabinet during operation of sequence (1). Subsequently, in sequence (5) the entry and exit action of `S3` are executed. After this point, the Process Model waits for three seconds as defined and, consequently, there is an indeterministic decision point in the Environment Reconfiguration Model where either an emergency is signaled up or not. We assume that the simulation determines to generate the emergency such that in sequence (6), the driver is called and the respective signal is produced. In sequence (7), the Adaptation Model receives this signal and switches to the emergency mode after polling a new event. Afterwards, the simulation starts validating whether the robot correctly drives to the emergency stop.

V. Implementation and Experimental Environment

Syntax and semantics of all used models were implemented in our *Model-driven Adaptivity Test Environment* (MATE). It provides an EMF (Eclipse Modeling Framework [11]) based metamodel and a simulator that can be used to execute the model automatically or—in order to debug it—step-wise.

In our previous work, we developed the Smart Application Grid (SMAG) framework that can be used for architectural run-time adaptation [12]. Based on SMAG, we created the self-adaptive HomeTurtle software. An impression of the physical

experimental environment is given in Figure 10. In order to show the feasibility of our validation approach, a platform-specific HomeTurtle test driver was developed as well. It directs the operation calls produced by the model to the real system and—vice versa—generates events from the system's observed behavior. However, not every modeled operation can be performed automatically. The initial configuration of the environment (setting up the cabinet's content, placing obstacles, etc.) and the validation whether the correct item was collected are performed manually by the test engineer. During the automatable phases, the validation directly benefits from the model-driven nature of our approach, its advantage in manually performed action is given by the reproducability of simulation paths. If any path fails during a test, it can be recorded, analyzed and later even be re-executed.

VI. Related Work

Validation approaches for self-adaptive systems are still rare in literature. An advanced strategy was proposed within the DiVA project [13]. The validation of DiVA-based implementations can be performed in two phases: (1) In the early phase, instances of the context model are generated and associated with partial solutions. Those describe how parts of the systems have to be configured after a certain context instance was applied and the corresponding adaptation was performed. (2) In an operational validation phase, the system's behavior is investigated during a sequence of contextual changes. The DiVA validation methods neither consider any system/adaptation interaction, nor do they propose specific test models.

Nehring and Liggesmeyer proposed in [14] a process for testing the reconfiguration of adaptive systems. The validation is performed in six iterations: In the beginning, a system model is derived and representative workload is prepared by a domain expert and later executed by developers or system engineers. In the second iteration, a system architect checks if structural changes are performed correctly. Thereby, the reconfiguration actions have to be in the correct order such that the system ends in a valid state and the quality of service is only affected minimally during reconfiguration. The third iteration considers data integrity while stressing the system with increasing load. In the fourth iteration step, state transfer between replaced components is investigated. An interaction issue between system transactions and the adaptation is tested in the fifth iteration. The last iteration considers the identity of components and component types before and after adaptation. In comparison to our approach, Nehring and Liggemeyer assume the adaptive system to be component based and the validation can be sufficiently investigated by a debugger-like tool chain. Thus, their approach is exploratory and hard to use for integration and system testing.

VII. Conclusions and Future Work

In this paper, we presented a concept to build black box simulation models for validating SAS. Our models are based on UML class models, state charts plus equivalence class trees with invariants. Automatons communicate by events such that the different concerns of the system process and adaptation can be separated. Our approach does not rely on any design model such that engineers are able to build discrete simulation models of arbitrary self-adaptive systems. The methodology comprises a process of classifying environment variability and defining an explicit model on its change. Using this toolset, we match the requirements (2)-(5) as stated in Section III. Requirement (2)–*Correct adaptation initiation* is considered by letting Adaptation Models receive signal events from the Environment Reconfiguration Models. Thus, the change in context can be causally connected with an adaptation of the system. As Adaptation Models define an operational order of adaptation actions, goal (3)-*Correct adaptation planning* is dealt with. Requirement (4)-*Consistent adaptation/system interaction* can be validated as the Process Model accesses the state of the Adaptation Models and defines conditions on this state. Thus, the system's adaptive behavior can be defined. As Adaptation Models can also check an adaptation's outcome by assertions, requirement (5)-*Correct adapation execution* is addressed.

In our future work, we are going to enrich the employed formalism (i.e., state charts, equivalence class trees, etc.) for more compact definitions and experiment with more complex scenarios in order to expand the evaluation. Concerning the improvement of formalism, for instance, we consider using Petri nets as they are more flexible in describing parallelism and synchronization, which is especially important when multiple widely-independent system parts interact.

Acknowledgment

This work is funded within the projects #100084131 and #100098171 (VICCI) by the European Social Fund as well as CRC 912 (HAEC) and the Center for Advancing Electronics Dresden (cfaed) by Deutsche Forschungsgemeinschaft.

References

[1] B. H. C. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in Dagstuhl Seminar 08031 on Software Engineering for Self-Adaptive Systems, 2008, pp. 1–26.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[3] IABG, "V-Modell XT 1.4," http://v-modell.iabg.de, visited 04/01/2014, 2012.

[4] M. Utting and B. Legeard, Practical model-based testing: a tools approach. Morgan Kaufmann, 2010.

[5] G. Püschel, S. Götz, C. Wilke, and U. Aßmann, "Towards Systematic Model-based Testing of Self-adaptive Software," in Adaptive, 2013, pp. 65–70.

[6] "TurtleBot 2," http://turtlebot.com, visited 04/01/2014.

[7] Object Management Group (OMG), "UML Specification, Version 2.4.1," http://www.omg.org/spec/UML/2.4.1/, visited 04/01/2014.

[8] Object Management Group (OMG), "Object Constraint Lanugage, Version 2.3.1," http://www.omg.org/spec/OCL/2.3.1/, visited 04/01/2014.

[9] M. Grochtmann, "Test case design using classification trees," Proceedings of STAR, vol. 94, 1994, pp. 93–117.

[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented Domain Analysis (FODA) Feasibility Study," DTIC Document, Tech. Rep., 1990.

[11] "Eclipse Modeling Framework Project," http://www.eclipse.org/modeling/emf/, visited 04/01/2014.

[12] C. Piechnick, S. Richly, and S. Götz, "Using Role-Based Composition to Support Unanticipated , Dynamic Adaptation - Smart Application Grids," in Adaptive, 2012, pp. 93–102.

[13] A. Maaß, D. Beucho, and A. Solberg, "Adaptation Model and Validation Framework – Final Version (DiVA Deliverable D4.3)," https://sites.google.com/site/divawebsite, visited 02/01/2014, 2010.

[14] K. Nehring and P. Liggesmeyer, "Testing the Reconfiguration of Adaptive Systems," in Adaptive, 2013, pp. 14–19.

 ISBN: 978-1-61208-341-4

A First Step Towards a Dependability Framework for Smart Environment Applications

Ehsan Ullah Warriach, Tanir Ozcelebi, Johan J. Lukkien
Department of Mathematics and Computer Science
Eindhoven University of Technical
Eindhoven, The Netherlands
{e.u.warriach, t.ozcelebi, j.j.lukkien}@tue.nl

***Abstract*—Smart environments will consist of a large number of heterogeneous devices that communicate to collaboratively perform various tasks for users. We propose a novel dependability framework to increase availability and reliability of smart environment applications. We argue that the key step in achieving high dependability is to predict faults before they occur. Many statistical fault prediction techniques have been proposed for smart environment applications. Selecting the best one among these techniques involves performance assessment and detailed comparison on given metrics. We present a linear regression-based prediction model to predict the remaining battery lifetime of a device to prevent faults due to low battery. Further, we discuss the proposed dependability framework, the basic approaches and the corresponding mechanisms to achieve our long-term research goal. We envision that dependability framework will reduce maintenance costs of large-scale smart environments and increase the dependability of smart environment applications.**

Keywords-smart environments*; *dependability*; *fault-prediction*; *battery fault-prediction model*; *linear regression.

I. INTRODUCTION

A smart environment is a physical space enriched with embedded Information Communications Technology (ICT) and adequate software modules that can communicate their local states, which are adaptive. From a technology point of view, sensor and actuator technologies, as well as communication standards are the main drivers for the development of today's smart environments. A convergence of these technologies raised interest in the smart environment research and its applications such as smart buildings (homes or offices), intelligent lighting, and remote health monitoring [1]–[3].

In smart environments, low capacity sensor and actuator nodes play an important role as they provide the bridge between the digital world and the physical world. A smart environment application relies first and foremost on sensory data acquired from multiple sensors in various locations of the real-world [4]. Sensor nodes are typically small, inexpensive, wireless, and battery-powered devices, prone to faults due to internal and external influences, such as low battery, miscalibration, hardware or software failures, environmental interferences and sensor aging. We define a *fault* as a deviation of at least one characteristic property or parameter of the system from normal operation. Faulty sensors deliver incorrect information to the application and this may lead to incorrect conclusions and consequently application failures, since sensors are usually left unattended for long periods of time in the field. Therefore, the adoption of smart environments is largely hindered by the fact that there is constant need for human (or even expert) intervention and the cost of maintenance of such systems is very high. Thus, dependable systems are required, evolving at runtime to maximize the availability and reliability of their applications. We identify two levels of dependability mechanisms, i.e., proactive mechanisms in the absence of faults and reactive mechanisms in the presence of faults. Systems that have the ability to identify faulty behaviors and make the necessary alterations to restore normal operation without human intervention [5] by means of a reactive dependability mechanism, such as fault tolerance through hardware redundancy, are said to be *self-healing*. On the other hand, systems that utilize proactive dependability mechanisms aim to predict and prevent faults before they occur, or at least delay them. There are two goals of this: *i)* to maintain application functionality as long as possible, and *ii)* graceful degradation of application performance. Fault prediction is required for proactive dependability and is the focus of this paper. It is a key mechanism of the dependability framework proposed in this paper, along with other mechanisms for fault monitoring, adaptation, fault tolerance, fault healing and fault notification.

Several fault prediction models for smart environment applications have been proposed in the literature. However, further research is needed to assess the quality and the resource requirements (e.g., memory, Central Processing Unit (CPU)) of these models. This paper describes work in progress for comparing various models for fault prediction against our proposed linear regression model. Linear regression analysis is one of the most widely used multivariate analysis methods, which assumes linear relationships between independent and dependent variables [6].

Faults can occur due to many reasons. For example, the battery is a critical resource of a battery-powered sensor, and it is one of the most common sources of faulty behavior [7] [8]. A battery powered node may start transmitting faulty values due to low battery [9] [10]. Predicting the remaining battery lifetime can help to use it more efficiently and to predict when a fault is likely to occur, allowing to take actions to prevent it. The need for reliable and accurate battery lifetime prediction

 ISBN: 978-1-61208-341-4

models have been expressed repeatedly in the literature. A battery depletion prediction model was introduced by Kevin et al. [11], where the Received Signal Strength Indicator (RSSI) value is monitored to predict the battery lifetime of sensor because RSSI becomes very low shortly before the depletion of the node battery. Profiling of the battery usage offline to make online predictions was proposed by Wen et al. [12], where the history of average energy consumption rate is used to predict the remaining battery lifetime of mobile devices. Takahashi and Ide [13], proposed a prediction model that uses previous battery usage pattern in the regression function as a trajectory in a feature space to predict the remaining battery lifetime.

In our approach, the discharge rates of running applications on a sensor network are modeled offline and this model is then used to predict the remaining battery lifetime online. We identify the Battery Dependent Components (BDCs) of a sensor that affect energy discharge rate. For example, radio (Transmitting (TX)/Receiving (RX)), Light-Emitting Diodes (LEDs), CPU, and sensor board are prominent BDCs. According to a regression model, remaining battery lifetime is a dependent variable, and energy consumption states of BDCs are independent variables. Our goal is to predict the remaining battery lifetime and take actions to avoid upcoming faults. In our future work, the considered fault prediction models will be evaluated based on a comparison of their resource requirements as well as *precision* and *recall* performance metrics.

The remainder of this paper is structured as follows. We propose the dependability framework for smart environments applications in Section II. The fault-prediction model is presented in Section III to predict the remaining battery lifetime of a sensor node using a linear regression model. The metrics that enable measuring the performance of a fault-prediction model are reviewed in Section IV. Finally, conclusion and future work are presented in Section V.

II. Dependability Framework for Smart Environments

Our long-term research goal is the development of a detailed dependability framework and to evaluate reliability and availability of smart environment applications as a result of self-x (self-protection, self-adaptation, self-healing) properties of the framework. Fig. 1 shows the high-level architecture of the proposed dependability framework, consisting of three main states.

The *fault prevention* state tries to predict faults and prevent the faults (proactive) by adapting the system and the applications based on available resources. The *failure prevention* state attempts to keep the system and the applications functioning with the help of fault tolerance and fault healing mechanism in the presence of detected faults (reactive). The *application failure* state notifies the system administrator to take mandatory actions against the detected fault to bring the system back into a safe state. Main mechanisms of the dependability framework are *i)* fault prediction, *ii)* adaptation, *iii)* fault monitoring, *iv)* fault tolerance, *v)* fault healing and *vi)* fault notification. Fault prediction looks to the future. It is based on monitoring the current state of a system in terms of resource attributes and also considers a history of such state information. Adaptation goal is to prolong the time before either a system or an application reaches a faulty state. Fault monitoring is responsible for monitoring and detecting violations of regular operating constraints (fault) in all hardware, software, and network configurations, as well as identifying the fault type. Fault tolerance refers to the ability of a system to avoid application failures in the presence of faults. Fault healing is the ability of a system to repair, update, or replace the faulty part. After healing, the system returns to a safe state. Fault notification is responsible to notify the user about the fault in a way that it causes minimal disruption to the user activity when it is not healed.

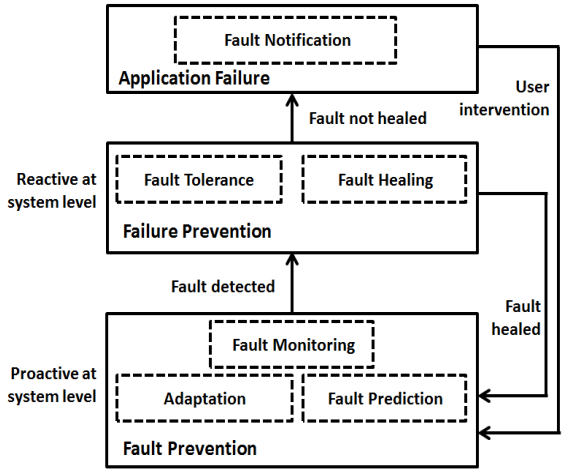


Figure 1: Dependability Framework for Smart Environments

III. Fault Prediction

In this initial phase of our work, we concentrate on the fault prediction model of the proposed framework. In general, given a fault prediction model, the system periodically monitors and logs the current state of the system at run-time and predicts the next state(s). In a smart environment, a set of resource attributes $Y = \{Y_1, ..., Y_K\}$ (e.g., battery levels or memory statuses of devices) are monitored and logged. Further, a number of statistical features $f \in F = \{F_1, ..., F_M\}$ are extracted from the history and the current value of $y \in Y$. The elements of f (e.g., minimum, maximum, expected value, gradient, mean, median, variance) are used in the fault prediction analysis. When a particular resource attribute of a device or the statistical features that correspond to the device are beyond the acceptable range (e.g., defined by thresholds of battery level), the fault prediction invokes the adaptation mechanism.

A fault prediction is defined by the triple $\{F_{type}, t_{PF}^{min}, t_{PF}^{max}\}$, where F_{type} refers to the type of the predicted fault, and $[t_{PF}^{min}, t_{PF}^{max}]$ refers to the interval in which the fault is expected. This is visualized in Fig. 2, where time t refers to a monitoring instance and $\triangle t_{WS}$ indicates how much into the past the fault

Figure 2: Fault Prediction Model Quality

prediction looks. $\triangle t_{LPF}$ is the time until the predicted fault interval. The prediction indicates that the fault of type F_{type} will occur in a time interval of length $\triangle t_{PF} = t_{PF}^{max} - t_{PF}^{min}$. A smaller value of $\triangle t_{PF}$ indicates a more accurate fault prediction model. If $\triangle t_{PF}$ is too large, it is very likely that a predicted fault falls within this determined interval. However, in this case the fault prediction is not useful since the fault can happen anywhere in the large interval.

A. Battery Fault Prediction Model

In this section, we introduce a prediction model for the remaining battery lifetime of a device. As shown in Fig. 3, the prediction of remaining battery lifetime can be divided into two main parts, namely, *offline modeling* and *online prediction*.

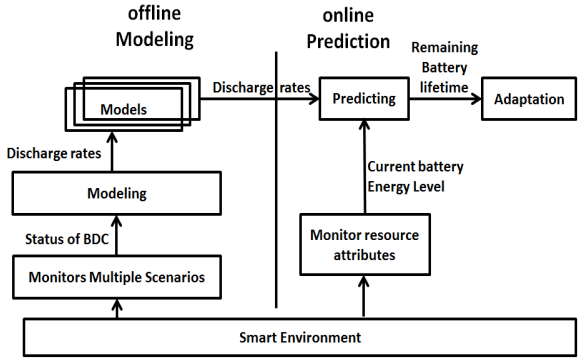


Figure 3: Battery Lifetime Prediction Model

Since the battery discharge rate varies according to the energy consumption of running applications (of BDCs that take a role), battery lifetime is application dependent. We identify a number of BDCs and their possible states, e.g., CPU (active, idle, standby), LEDs, sensor board and radio (TX/RX). The radio component can have different settings of TX and RX modes based on either available battery of a device or application specific. We quantify the relation between BDC states and the battery discharge rate using a linear model, resulting in a multiple linear regression model that employs application specific battery discharge rates. Whenever the application behaviour changes, the fault prediction model needs to be revised using a multiple linear regression model to calculate the current battery discharge rate. These are then used during the *online prediction* process together with the current battery energy level to predict the remaining battery lifetime.

B. Linear Regression Model

Linear regression model [6] has been successfully used for forecasting and prediction in various fields and we consider this model to predict the remaining battery lifetime. Multiple linear regression models the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data [6].

Consider a set of tasks $\tau = \{\tau_1, ..., \tau_N\}$ running on a device, where each task has a battery discharge rate $R = \{R_1, ..., R_N\}$ based on the BDCs energy consumption states $X = \{X_1, ..., X_P\}$ while running that task. The external or internal events can influence the running task and change the state of a BDC. For example, a task can dynamically pick one of the data sampling periods T_1 and T_2, specifying that its battery discharge rate also varies dynamically. Therefore, we need to identify the quantitative relationship between different states of the task and their discharge rate. We consider the states of BDCs as independent variables (X for explanatory) and battery discharge rate as dependent variables (R for response). Linear regression is used to predict the values of the response variables, $(r_1, ..., r_N) \in R$, given a set of explanatory variables $(x_1, ..., x_P) \in X$ (states of BDCs). The relationship between the explanatory variables and the response variables is given by the following equation 1:

$$r_N = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12} + ... + \beta_p X_{1p} + \varepsilon_i \quad (1)$$

where,

$$R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix}, X = \begin{pmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_N^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1P} \\ x_{21} & \cdots & x_{2P} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NP} \end{pmatrix}$$

$$\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_P \end{pmatrix}, \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_P \end{pmatrix}$$

- R is a $(N \times 1)$ dependent variable matrix, where N is the number of tasks (discharge rates).
- X is a $(N \times P)$ matrix of independent variables, where $x_{n,p}$ is the state of the p^{th} BDC while task τ_n is running.
- β is a $(P \times 1)$ vector of regression coefficients.
- ε is a $(N \times 1)$ vector of additive random error. We assume that the error ε_i is a statistical error, which is normally distributed with mean zero and variance σ^2, abbreviated as $N(0, \sigma^2)$ [6].

The linear regression function of the battery discharge and BDCs is described using the equation 2:

$$E(t) = e_0 - \beta * t \quad (2)$$

where, e_0 and $(-\beta)$ are the intercept and slope of the line respectively. $\bar{e} = \frac{\sum_{i=1}^{n} e_i}{n}$, $\bar{t} = \frac{\sum_{i=1}^{n} t_i}{n}$, $\beta = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(e_i - \bar{e})}{\sum_{i=1}^{n}(t_i - \bar{t})^2}$ and $e_0 = \bar{e} - \beta * t$, β measures the change in the mean of E for a unit change in t, which is the discharge rate of the battery [6]. In order to explain the battery lifetime prediction model, let us suppose we have a set of data samples $(t_i, e_i), i = \{1, 2, ..., n\}$, as shown in Fig. 4.

It shows the prediction of the battery lifetime of a sensor, where e is the value of battery and t is the time. Suppose,

 ISBN: 978-1-61208-341-4

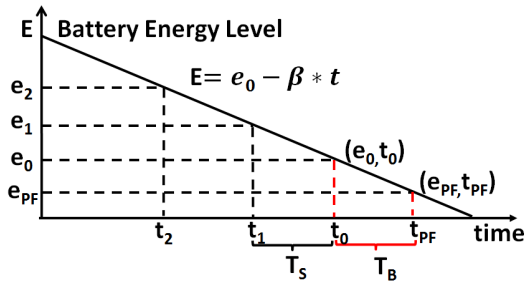


Figure 4: Estimation Curve of a battery discharge rate

a system monitoring module measured the battery energy level e_0 at the time t_0, which is beyond the acceptable range. Then, battery lifetime prediction model is invoked to determine the remaining battery lifetime against the current status of BDCs states and using the current battery energy level e_0. Consequently, we can have an estimation of remaining battery lifetime using the discharge rate. We assume that the battery energy level is e_{PF} at time t_{PF}. Then, by observing Fig. 4 and by applying following Equation 3, we can measure the remaining battery lifetime:

$$T_B = t_{PF} - t_0 = \frac{e_0 - e_{PF}}{\beta} \quad (3)$$

IV. Evaluation of Fault Prediction Model

In order to investigate the quality of fault prediction model and to compare various fault prediction models against our proposed linear regression model it is required to identify suitable metrics. The goal of a fault prediction model is to predict faults accurately, efficiently and in a timely manner. An accurate fault prediction model would accomplish a one-to-one matching between predicted and true faults. A fault prediction is a *True Positive* (TP), if a fault occurs within the predicted period. If no fault occurs and a fault is predicted, the prediction is a *False Positive* (FP). If the model misses to predict a true fault, it is a *False Negative* (FN). If no true fault happens and no fault notice is given, the prediction is a *True Negative* (TN). Further, we consider precision and recall based on above metrics. *Precision* is defined as the ratio of correctly identified faults to the number of all predicted faults $precision = \frac{TP}{TP+FP}$. *Recall* is the ratio of correctly predicted faults to the number of true faults $recall = \frac{TP}{TP+FN}$ [14]. *Accuracy* is the number of correct predictions over the total number of predictions made. Further, we will investigate the *computational* requirements of fault prediction models, e.g., memory and CPU.

V. Conclusion and Future Work

We presented a high-level architecture of a dependability framework for smart environment applications. In this context, we consider the problem of predicting faults in smart environments. As a first step, we presented a prediction model based on the multiple linear regression model for the remaining battery lifetime of a device. In order to develop an accurate and efficient fault prediction model, we must understand the trade-offs among the metrics defined in Section IV for each prediction model and choose the best tradeoff for a given dependable framework for smart environments. Our future work will focus on implementing this architecture on top of an operational platform in a real smart environment in order to guarantee the availability and reliability of applications. The ability of applications in a system to survive free of faults depends on adaptations supported by the dependability architecture. We recognize that the system may be affected by many types of faults. Thus, we will investigate optimal application adaptation mechanisms as well as fault prediction models against other types of faults, e.g., low memory, link quality, hardware or connection failures, miscalibration. Our ultimate goal is to develop a dependability framework for smart environments to provide users with services, which are highly reliable and available.

Acknowledgment

This work has been supported by the ProHeal project (n. 10017751) funded by the Information Technology for European Advancement (ITEA2).

References

[1] D. Cook and M. Schmitter-Edgecombe, Assessing the quality of activities in a smart environment, Methods of Information in Medicine, vol. 48, no. 5, 2009, pp. 480-500.

[2] P. Yu, X. Ma, J. Cao and J. Lu, Application mobility in pervasive computing: A survey, Pervasive and Mobile Computing, vol. 9, no. 1, 2013, pp. 2-17.

[3] S. Bhardwaj, T. Ozcelebi, O. Ozunlu, and J.J. Lukkien, Increasing reliability and availability in smart spaces: A novel architecture for resource and service management, IEEE International Conference on Consumer Electronics (ICCE), 2012, pp. 439-440.

[4] F. L. Lewis, Wireless Sensor Networks: Smart Environments, John Wiley & Sons, Inc., 2005, pp. 11-46.

[5] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya, Self-healing systems - survey and synthesis, Decision Support System, vol. 42, no. 4, 2007, pp. 2164-2185.

[6] L. Wasserman, Lecture Notes for Linear Regression, 2010, Retrieved 08-04-2014. [Online]. Available: http://www.stat.cmu.edu/ roeder/stat707/lectures.pdf.

[7] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, Lessons From A Sensor Network Expedition, European Conference on Wireless Sensor Networks, 2004, pp. 307-322.

[8] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, Wireless sensor networks for habitat monitoring, 1st ACM international workshop on Wireless sensor networks and applications, vol. 2, 2002, pp. 88–97.

[9] E.U. Warriach, M. Aiello, and K. Tei, A Machine Learning Approach for Identifying and Classifying Faults in Wireless Sensor Network, IEEE 15th International Conference on Computational Science and Engineering (CSE), 2012, pp. 618-625.

[10] N. Kevin et al., Sensor Network Data Fault Types, ACM Transaction Sensor Network, vol. 5, no. 3, 2009, pp. 25:1–25:29.

[11] I. H. Yano, V. C. Oliveira, E. A. de Mello Fagotto, A. A. Mota, and L. T. M. Mota, Predicting battery charge depletion in wireless sensor networks using received signal strength indicator, Journal of Computer Science, vol. 9, no. 7, 2013, pp. 821-826.

[12] Y. Wen, R. Wolski, and C. Krintz, Online Prediction of Battery Lifetime for Embedded and Mobile Devices, Power-Aware Computer Systems - Lecture Notes in Computer Science, vol. 3164, 2005, pp. 57-72.

[13] T. Takahashi, and T. Ide, Predicting battery life from usage trajectory patterns, 21st International Conference on Pattern Recognition (ICPR), 2012, pp. 2946-2949.

[14] F. Salfner, M. Lenk, and M. Malek, A Survey of Online Failure Prediction Methods, ACM Computer Survey, vol. 42, no. 3, 2010, pp. 10:1–10:42.

 ISBN: 978-1-61208-341-4

ContextPoint: An Architecture for Extrinsic Meta-Adaptation in Smart Environments

Christian Piechnick, Sebastian Richly, Thomas Kühn, Sebastian Götz, Georg Püschel and Uwe Aßmann
Software Technology Group, Technische Universität Dresden,
Dresden, Germany
Email: {christian.piechnick, sebastian.richly, thomas.kuehn3, sebastian.goetz1, georg.pueschel, uwe.assmann}@tu-dresden.de

***Abstract*—The establishment of mobile devices had a high impact on the use and development of software systems. It is expected that the ability to automatically adapt to changing environments will be a crucial property for future apps running on mobile devices. The problem with current approaches for self-adaptive systems is that developers must define the adaptive behaviour at design-time. In many cases, however, the developer cannot predict all situations at design-time, which should trigger adaptation at runtime. Furthermore, applications for mobile devices are usually optimized for a small set of use cases and have a narrow, well-defined scope. In order to support more complex tasks, the functionality of several apps has to be composed dynamically. Another problem arises from the ever increasing number of available applications. In this paper, we address these problems by proposing a novel infrastructure for self-adaptive systems in smart environments, namely ContextPoint. Our goal is to describe an architecture which supports unanticipated adaptation for single systems, as well as the automatic integration of actuators and sensors, situated in the environment, with services and data from both, mobile devices and the cloud. Therefore, a distributed adaptation technique is proposed, where adaptation logic and rules are provided by the environment itself. This decentralization simplifies the development of self-adaptive systems with dynamic adaptive adaptation processes (meta-adaptation) and, thus, the design and operation of systems with unanticipated adaptation. Furthermore, our approach provides means for describing context-dependent collaboration between varying systems enabling the design of ad-hoc system-of-systems.**

***Keywords*—Adaptation; Self-Adaptive; Meta-Adaptation; Architecture; Context-aware; Location-aware.**

I. INTRODUCTION

The wide-spread acceptance of mobile devices (e.g., smartphones, tablets) changed the development as well as the use of software applications radically. Because applications running on mobile devices change their location and, hence, their environmental situations they are used in, they have to adapt their appearance and behaviour accordingly. This kind of flexibility is commonly called context-aware adaptation in self-adaptive systems. Currently, such systems rely on an environmental model (i.e., context model), which describes the entities of the execution context and their relationships. In these models, software engineers predefine statically at design-time which contextual information can be observed at runtime. At runtime a MAPE-K-Loop [1] (a) *monitors* the environment using sensors, (b) *analyses* the gathered data to instantiate the context meta-model, (c) *plans* necessary reconfigurations, and (d) *executes* the chosen plans. The main problem, however, is that software developers usually cannot predefine all environmental entities, which could be important for an adaptation process at runtime, at design-time. Lets consider an application that mutes a smartphone automatically, every time the user must not be disturbed (e.g., the user is participating in a meeting). The fact, however, that a user would be disturbed by a ringing smartphone is highly individual. A developer of such an application could most possibly not foresee all individual cases (e.g., when a baby is sleeping within the same room the user is located). To address this problem, the adaptation process itself should adaptable.

Another consequence arising from the characteristics of mobile devices is the change in size and range of functions. Traditionally, software system for stationary devices increased in their code size and complexity. The goal was to create multi-purpose systems with a huge set of provided functionality. Applications for mobile devices, henceforth denoted as *apps*, reversed that trend. They usually have a narrow scope with a rather small set of offered functionality. Those apps are optimized for a well-defined set of tasks. We call this *Functional Separation of Concerns* (F-SoC). In order to support more complex workflows, several apps have to work together to combine their provided functionality in a seamless way. Currently, there is no mechanism to describe an overall workflow across multiple apps on mobile platforms. On Android, for example, it is only possible to exchange data between applications using intents. Intents restrict an application' access to another application's provided services or data. Those intents are specified using coarse-granular classes of access types defined in the system frameworks. The problem is that developers would need to agree on a shared set of guidelines (like datatypes) in order to establish a seamless integration, which is hard to enforce for domain-specific applications from different domains. Through the establishment and spread of devices for smart environments, this kind of functionality becomes even more important. Since sensors and actuators become cheaper and more standardized, they can be placed easily in any kind of environment. It is likely that, in the future, users want to integrate their apps on multiple mobile devices seamlessly with services provided by both the environment and via Internet, depending on their current situations.

The main problem resulting from the F-SoC expansion is the huge number of similar apps developed for different tasks or usage-scenarios. The Google Play Store contain almost one million apps each. After the user found an appropriate app in this huge variaty of offers, it has to be installed manually and, in many cases, be configured. In order to increase convenience and efficiency, regarding the usage of mobile devices, it should be possible to automatically detect a set of apps that are well-suited to support a user's task in a given situation and automatically deploy, configure and connect those applications.

The mentioned requirements for context-aware, mobile app-infrastructures can be summarized as follows. Apps have to support:

R1 **unanticipated adaptation** by *meta-adaptation*, i.e., the adaptation mechanisms need to be adaptable

 ISBN: 978-1-61208-341-4

themselves, because the developer cannot foresee all possible situations the application will operate in.

R2 **runtime composition**. In order to support complex tasks, apps have to be combined to ad-hoc systems of systems (SoS). Those apps can either be executed on the same device or be integrated as services in a distributed system.

R3 **automatic provisioning**. Based on the context and the task of a user, a collection of suitable apps have to be provided.

In this paper, we present an approach which satisfies all three requirements by introducing the concept of extrinsic meta-adaptation in a self-adaptive control loop. Usually, the mechanisms required for self-adaptive systems are implemented statically. Even though, many approaches propose architectures to support adaptive monitoring and analysis by distributing those steps to a varying network of collaborating systems, *plan* and *execute* are usually fixed in their implementation. Even though every self-adaptive systems relies on a component model that can theoretically be extended at runtime, currently there is no process how to dynamically extend the knowledge base used for adaptation as well as how to adapt the adaptation process itself. Furthermore, systems with a flexible monitor and analyze phase are able to dynamically extend the information sources that are used for decision making, while the strategies how those information are processed remain fixed. A self-optimizing system for non-functional properties for example, might at runtime extend the information required to perform optimization and include new components that can be used by the planning component, but will still only optimize non-functional properties. In situations where it is necessary to adapt for self-healing or functional requirements, such a system will fail. Meta-adaptation allows to adapt the adaptation process itself (e.g., introduce new plan and execute logic, etc.) at runtime. We want to propose an adaptation mechanism where apps on mobile devices can automatically be adapted, connected, and provided from an environmental infrastructure. Because the meta-adaptation is provided by the environment itself (i.e., extrinsic), the overall adaptation process can be extended at runtime by changing the location without redeployment of the entire self-adaptive system.

This paper is structured as follows: In Section II, we give a short introduction to the Smart Application Grids (SMAGs) approach, which is used as a basis for the approach presented in this paper and introduce the concept of meta adaptation in Section III. We discuss our approach in Section IV. Section V provides an evaluation of the presented concept using an example. In Section VI, related work is presented. Finally, Section VII presents our conclusion and future work.

II. Smart Application Grids

In order to dynamically adapt an application to varying contexts, the structure and behaviour of an application has to be changed at runtime. Hence, the application architecture has to be *variable* and *extensible*. Variability enables the adaptation of existing behaviour at runtime within a given variability space. In contrast to that, extensibility allows to scale the variability space and build the foundation for Meta Adaptation.

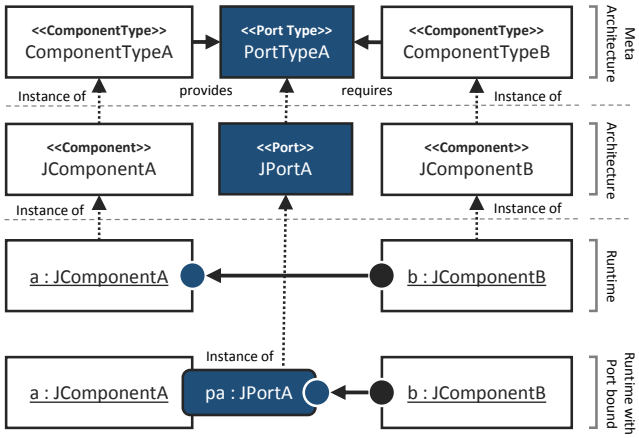


Fig. 1. The Meta-Levels of Smart Application Grids

Traditionally, variability is assured by applying the *Template Hook Meta Pattern* [2]. The application logic is separated in a fixed (template) and variable part (hook). By exchanging the hook at runtime, an application's behavior is adapted dynamically. This very simple procedure introduces three problems. First, the replacement of the complete hook can be expensive concerning resource usage and replacement time. Second, the system might be in an inconsistent state during reconfiguration. Third, if the hook is stateful, the state has to be migrated to the new hook, which can be expensive as well. Because in many designs, the hook can also have external references, both incoming and outgoing, those references have to be migrated, too. To tackle those shortcomings, invasive composition techniques (i.e., Aspect Oriented Programming) for dynamic context-aware adaptation were investigated [3]. With runtime weaving it is possible to exchange program code in a very fine-grained manner during the lifetime of an application. Still, aspects introduce some problems as well. First, it is a code-composition technique on meta-layer M1 [4], i.e., the class-layer. Consequently, it is only possible to change the behaviour of all objects instantiated by a given class [5]. However, for many scenarios it is necessary to have an adaptation technique on meta-layer M0, i.e., the object-layer, which allows to change the behaviour of individual objects. Another major drawback of aspects is that they are a white-box composition technique (i.e., aspects rely on the internals of the application subject to adaptation) which decreases reusability. Consequently, reusing adaptive behaviour across different applications and domains is insufficiently supported. In order to support invasive software composition on an architectural (component) level, we developed our Smart Application Grids (SMAGs) framework. SMAGs is a model-driven, platform independent design and operation principle for fine-grained, dynamic and unanticipated adaptation with a focus on increased reuse. SMAGs consists of many small, distributed applications that are linked dynamically. Role-Based Design and Programming [6] is used to change the structure and behaviour of individual applications as well as to express dynamically varying relationships across several distributed applications. A *role* is a dynamic service of an object in a specified context, offering the possibility to express separation of concerns,

 ISBN: 978-1-61208-341-4

interface-structuring, dynamic collaboration description, and access restriction [7]. Roles are played by objects, dynamically altering the structure and behaviour of the player. In other words, roles enable an adaptation technique on meta-layer M0, the object-layer. On the one hand, roles can extend the object's state and functionality (i.e., introduce new methods and/or attributes). On the other hand, the objects existing behaviour can be changed. Furthermore, roles contain references to other roles. Since roles can be played and discarded at runtime, they are capable of expressing dynamically changing relationships across multiple system entities (e.g., objects, components, etc.). The major difference to other invasive composition approaches (e.g., aspects) is that roles are played within a context (e.g., "a person plays the student role within the context of an university"). This tight coupling between behaviour and environmental conditions makes Role-Based Design a powerful approach to model *Self Adaptive Systems* (SAS).

As depicted in Figure 1, at design-time, a platform independent **Meta Architecture** defines *Component Types* which specify provided and required *Port Types*. A Port-Type represents an interface specification. From several Meta Architectures a platform specific architecture can be derived, describing *Components* implementing *Component Types* and *Ports* implementing Port-Types. Ports can be grouped into *Port Models*. Each Port Model is associated with a binding to components and with environmental conditions, stating when it should be integrated into the application. At runtime, Components can be instantiated and connected by their matching required and provided Port Types. With the instantiation of Ports/Port Models and their binding to Components, the behaviour of individual *Component Instances* as well as the structure of the overall application can be adapted according to a given context. For more detailed information we refer to [6].

The SMAGs approach proposes an adaptation architecture. In [6], an overview of this architecture is presented. It represents a *MAPE-K loop* with the Sensor Layer monitoring the environment and transferring the gathered information to a *Context Model*. An *Inference Layer* relates existing and deduces new information based on the data in the Context Model. An *Adaptation Layer* creates reconfiguration plans based on the information from the Context Model and the current configuration of the adaptive application. Those plans are then executed by a Runtime Environment. Because the adaptation architecture itself is a SMAGs-based application, the concrete implementations (e.g., the Context Model representation etc.) can be changed at runtime. This adaptive adaptation architecture enables Meta Adaptation.

Furthermore, the SMAGs approach is based on a distributed repository infrastructure. A repository can be used to store the meta-architecture and architectural information as well as component and port implementations. These artefacts can either be reused at design-time for the design and implementation of new systems or at runtime to extend a running application with new components and ports. Additionally, each repository exposes a *Service Trader*. Applications can register remotely to offered functionality alongside with contextual information at the Service Trader. Other applications can query the published services to autonomously create dynamically varying SoS. The SMAGs approach is used to model a novel adaptation paradigm for unanticipated adaptation in mobile scenarios.

III. Extrinsic Meta-Adaptation

Adaptation mechanisms for context-aware software system can be classified into *parametrised*, *control-flow based* and *compositional* adaptation [6]. For parametrised adaptation the application units expose predefined parameters that can be changed at runtime. Control-flow based adaptation triggers the execution of application-specific behaviour to react on environmental changes. Compositional adaptation allows to change the structure of the application (e.g., create, remove, or reconnect components, etc.). The available components as well as the provided composition operators define which variants of the system are valid configurations, constituting the *variant space*. The other key modelling element, is the adaptation strategy that describes *when* adaptation has to be triggered and *how* the system should be reconfigured in a given situation. Therefore, a context metamodel describes all types of contextual information that may be available at runtime. Whenever the concrete context model changes, the system checks whether or not one of the variants within the variant space is better suited than the current system configuration. When the system detects a better alternative, a reconfiguration plan is generated. However, the main problem for software developers is that they cannot foresee all possible conditions that should trigger adaptation as well as all other systems, with which the application might collaborate. When the variant space as well as the adaptation strategy is fixed, unplanned situations cannot be handled. In order to support adaptation in unanticipated situations the adaptation process itself must be variable and extensible. This concept of adapting the adaptation is called meta-adaptation [8]. Figure 2 shows how meta-adaptation can be achieved by extending (1) either the application's variability space or (2) the adaptation logic itself.

Variability-Space The variability space (Figure 2 top-right) defines which variants of the system exist. In theory, an adaptation process investigates all different alternatives to decide whether or not there is a better configuration of the system w.r.t. the current environmental conditions. The variability space is constructed using all possible component/port combinations and is constrained by architectural templates and rules. Adding new artefacts or changing existing ones changes the variability space at runtime (Extension). This allows to create new variants, not considered at design-time, satisfying dynamic requirements the developers could not foresee.

Adaptation Process When the implementation of a MAPE-K loop itself provides variability and extensibility, the adaptation loop itself can be reconfigured (Figure 2 bottom-right). In the SMAGs approach, the MAPE-K loop is implemented using the same composition system as the system it is adapting. By this design, it is possible to adapt the adaptation architecture, which enables meta-adaptation. Figure 2 shows a second MAPE-K loop that uses the Runtime Environment of the adapted application as an executor for the reconfiguration plan. The second loop can be implemented in any application. This allows, for example, to exchange the context model representation, to introduce parallel representations with different characteristics (e.g., probability) of the context model, to introduce new sensors, extend the inference mechanisms or introduce new planners and executors.

SMAGs supports both dimensions of meta-adaptation. The repository infrastructure enables applications to extend the

 ISBN: 978-1-61208-341-4

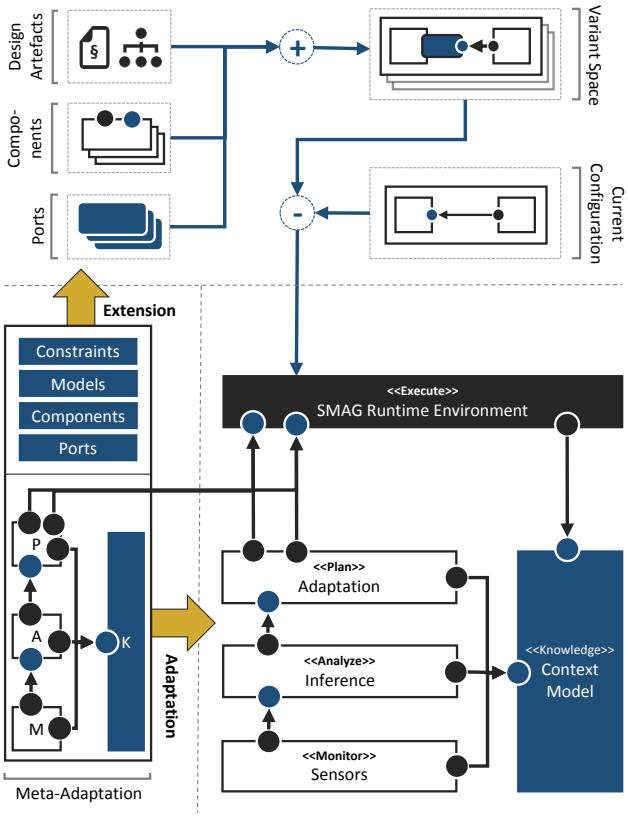


Fig. 2. The SMAGs Meta-Adaptation Process

variability space at runtime, by importing new model artefacts (i.e., (Meta) Architectures and Port Models) as well as implementation artefacts (i.e., Components and Ports). Furthermore, SMAGs supports runtime reconfiguration of the adaptation process because the MAPE-K loop is itself implemented as a SMAGs application. This enables the developer of meta-adaptive applications to change parameters of existing MAPE-K loop components as well as to change the structure of MAPE-K loops. Furthermore, the introduction and binding of new ports can extend or change the behaviour of existing components. This allows to dynamically adapt the context model representation (e.g., add the concept of uncertainty to specific model entries) and to bind according inference strategies.

These two dimensions build a foundation to create location-aware unanticipated adaptation by integrating meta-adaptive systems within the environment. Whenever a mobile application is situated in the same location, the meta-adaptive system can provide location-specific adaptation knowledge.

IV. The ContextPoint

In the research area of context-aware and self-adaptive systems, still no common definition of the term *context* was established. The most accepted and used definition was given by Anind K. Dey in 2001:

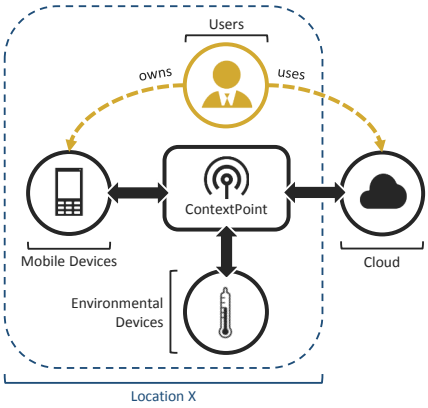


Fig. 3. The Context-Diagram of ContextPoint

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [9]

Over the last decades many conceptual frameworks for context modelling and interpretation were developed [10] [11] [12] [13]. One commonality is that every definition and conceptualization of the term context treats the location of both the user and the application as a first class citizen. Consequently, the location is a central entity of context-aware adaptation. Nevertheless, already in 1998, Schmidt et al. [14] observed that the exact physical location is only sufficient for a rather limited set of adaptation scenarios. The more important information for adaptation is the semantics of the location and the implications that can be reasoned about the fact that an application is located at a given place at a given time. Especially in indoor scenarios, where it is hard to determine the absolute position of an object, a symbolic representation of the location becomes very important. When, for example, a person enters a meeting room, where a meeting takes place at this time, it can derive that the person is participating in a meeting. Because indoor localization is still difficult and usually requires a special sensing infrastructure involving high costs and high setup efforts, this kind of location aware services could not gained wide spread acceptance.

The ContextPoint is a device that observes the environment and acts as a coordinator for several self-adaptive applications within this environment. The goal of ContextPoint is to provide an easy to install and easy to use infrastructure to enable context-aware adaptation based on the location of the user. Figure 3 shows a context diagram of ContextPoint with three types of interacting services: *Mobile Devices*, *Environmental Devices* and *Cloud Services* mediated by the ContextPoint. The basic principle for interaction is locality. When a user, carrying a mobile device, is close to a ContextPoint, information and services implied by the location and the task of the user are provided automatically.

In this section, we first outline the Top-Level Architecture of the ContextPoint. Afterwards, we explain the main features and their relation to the requirements, stated in Section I. Furthermore, we describe how the ContextPoint architecture relates to the concept of meta-adaptation. Finally, we outline

 ISBN: 978-1-61208-341-4

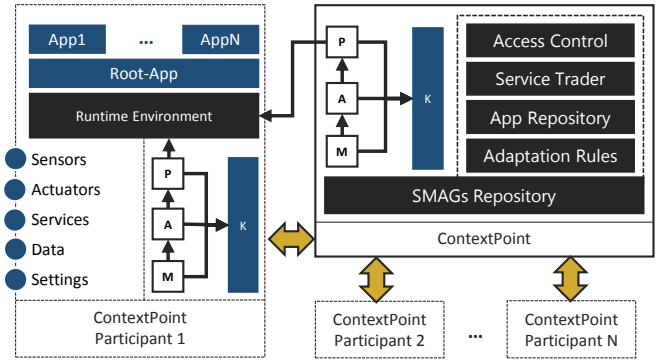


Fig. 4. The ContextPoint Architecture

the crucial aspect of privacy and security.

A. Top-Level-Architecture

ContextPoint is embodied as a device that is integrated in the infrastructure of an arbitrary location, providing contextual services by placing a mobile device at it. The ContextPoint device as well as the mobile devices initiate communication using Near Field Communication (NFC). Therefore, on the mobile device a *ContextPoint Root-App* is running in the background, receiving NFC-Events. When the device is placed near the ContextPoint, it recognizes the ContextPoint's signature and initializes communication. The ContextPoint shares local Wi-Fi credentials to enable the device to automatically connect for a long-term communication. After the device connected to the local Wi-Fi, the Root-App will register itself at the ContextPoint device which contains a SMAGs repository storing implementation artefacts, contextual information, and adaptation rules; offering a Service Trader as well as providing access control.

Figure 4 shows the Top-Level-Architecture of ContextPoint with *ContextPoint Participants* as an abstraction of mobile devices, environmental devices and cloud services which are mediated by the ContextPoint. Each participant runs a ContextPoint Root-App responsible for the communication with the ContextPoint and the provision of base services for Sensors, Actuators, Data, and Settings. This app acts as a runtime environment for all SMAGs-Apps deployed on the device. Each app can publish services and connect to services offered by other participants. The Root-App is responsible for the registration and de-registration at a ContextPoint. After registration, different capabilities can be integrated into the environmental infrastructure. The data delivered by the **Sensors** can be used as additional data sources to construct the ContextPoint context model. Simple services for manipulating its **Actuators** as well as more complex **Services** can be published at the Service Trader. In consequence, they can be used by other applications registered at the same ContextPoint. Furthermore, special services for accessing **Data** (e.g., user profile) or reading and manipulating the devices **Settings** can be published, to externally change the state of a device or service.

B. Location Specific Apps

In the *App Repository* of ContextPoint, location specific apps (e.g., a slide-presentation app for a meeting room) can be stored together with general metadata, contextual information and access rights. When a mobile device registers at a ContextPoint, the Root-App queries the App Repository and presents an overview of all available apps to the user. Based on the user's identity, this list might be filtered according to the access rights of the applications and roles of the user (e.g., speaker) and the current context. Furthermore, the list can be filtered and prioritized using contextual information (e.g., business applications) and application specific metadata using roles. The user can then select apps that are automatically installed within the Root-App. Together with the application, the ContextPoint device provides a context- and user-specific initial configuration (e.g., use the meeting room's projector as a default presentation device). This automatic App-Provisioning covers requirement **R3**.

C. Location Specific Settings

Each participant can expose read and write services for its settings to the environment. Alongside with a capability model of the participant, an access control component is generated by the Root-App. Based on the context, the user and the defined access rights, other devices as well as the ContextPoint itself can evaluate and change the settings of the participant (e.g., mute the mobile device). This kind of parameterized adaptation allows to change the state of a participant in unforeseen ways, addressing requirement **R1**.

D. Service Trader

SMAGs applications can offer a subset of their provided functionality for remote access (e.g., an interface to present and control a presentation on a smart projector). Those dynamically published services can be registered at the ContextPoint. Alongside with the structural description, a subset of the applications' context model, the identity and metadata of the owning user as well as the functionality can be published. Other participants can search for required services that are appropriate with respect to their own context (e.g., search for presentation services). This search is provided by the Service Trader **R2**.

E. Checkout

A participant can be signed off from the ContextPoint by either using a checkout service of the Root-App or when the connection to the ContextPoint is lost. SMAGs Repositories can be interconnected by a Peer-to-Peer network [6]. This infrastructure is used to notify ContextPoint devices when a participant signs in another ContextPoint. In order to avoid concurring adaptations, ContextPoints will close the connection to participants that are still signed in, but have been detected at another ContextPoint devices.

F. Meta Adaptation

In the SMAGs approach, the MAPE-K loop is modelled explicitly using a component architecture, whereby the flow between the components is modelled implicitly using events. As depicted in Figure 4, the ContextPoint runs a MAPE-K

 ISBN: 978-1-61208-341-4

loop that can adapt participants. Like any other SMAG-based control loop, the loop itself is a SMAGs application.

Consequently different adaptation strategies and context models can be used for different participants or situations. The ContextPoint gathers contextual information using sensors exposed by other participants (e.g., mobile devices, environmental sensors) and stores this information in its local context model. The execution layer of this control loop is the SMAGs runtime environment of the corresponding participant. This allows the ContextPoint to query a participants application runtime model to decide whether or not an adaptation is necessary. Since SMAGs supports parametrized, control-flow based, and compositional adaptation; all three adaptation mechanisms can be used to adapt the MAPE-K loop. This Meta Adaptation triggered by the environment supports adaptation that the developer initially did not foresee, which supports requirement **R1**. Since meta-adaptive SMAGs apps can dynamically connect several applications within one control-flow, also requirement **R2** is tackled.

G. Security and Privacy

We are aware that the proposed architecture creates serious security (e.g., abuse of devices) and *privacy* (e.g., unauthorized access to personal data) threats. On the one hand, the owner of a ContextPoint device must be sure that only approved participants can get access to the provided service- and data-infrastructure. On the other hand, a participant wants to make sure that private data cannot be accessed by other participants and that neither data nor services from potentially compromised sources are used. Because the software running on ContextPoint devices and the participants devices is realized by SMAGs applications, the role-based adaptation mechanism can be used for security and privacy adaptation. Currently, the following mechanisms are included: First, every ContextPoint device has at least one owner that can regulate which participants can sign in. By default two sign-in strategies are supported. Either the owner grants the access for all users, or he has to confirm each user. Second, in order to ensure client-side privacy, *Filter-Ports* [6] can be used to restrict access functionality offered by a component. Special *Access Ports* by default restrict any access to the underlying functionality, only granting access to those participants the user has defined. Hence, within the Root-App the user has the possibility to define which services can be used by which other participants. One serious threat is the possible abuse of the capabilities of Meta Adaptation. One way to address this issue is to use Access-Ports for the MAPE-K loop, too. Because the services of the runtime environment for querying the application model and executing reconfiguration scripts are SMAGs Ports, Filter-Ports can equally be used to restrict the access to the remote services of the runtime environment. When a user does not trust the ContextPoint at a given location he can force the application to not expose any information about the application architectures and forbid any remote access to the reconfiguration system.Security and privacy threats are important topics for adaptive systems in general, especially in extrinsic unanticipated adaptation. We argue that the role-based adaptation mechanism of SMAGs is a well suited mechanism towards safe and secure self-adaptive systems, which is to be investigated in detail in future work.

V. Implementation

To show that the proposed architecture concept for location-based extrinsic Meta Adaptation is feasible, we have implemented ContextPoint as well as several ContextPoint apps using the Java-based implementation of the Smart Application Grids runtime environment. As ContextPoint device, we used a Windows 7 notebook with a Standard JVM. In future, we plan to investigate the use of a Raspberry Pi [15] due to its smaller dimensions and lower energy consumption. On the notebook a ContextPoint application was running on top of the SMAGs runtime environment, supporting the features presented in Section IV. A USB NFC Reader was connected to the notebook for the initial sign-in procedure for NFC-ready mobile devices. As a mobile, device we used a Nexus 7 Android tablet with Android version 4.1.

Our sample scenario is based on a smart meeting room with a built-in, remotely controllable projector, a light system and the ContextPoint device. When a person enters the meeting room he holds his smartphone against the ContextPoint, which exchanges the local Wi-Fi credentials via NFC. The Mother-App running on the smartphone receives the ID of the ContextPoint and the Wi-Fi credentials. Afterwards the user is asked if the smartphone should login into the local Wi-Fi. After the user confirmed to log in, the Mother-App scans the local Wi-Fi for the ContextPoint with the given ID using the Universal Plug and Play (UPNP) protocol. When the Mother-App has found the corresponding ContextPoint it uses the registration API to authenticate and publish a description of available services. In this case the smartphone exposes a brightness sensor that can be used to determine the rooms brightness. The ContextPoint determines via its context model that a meeting is taking place in this room at the time the person enters the room. Based on a rule, the owner of the ContextPoint defined, the smartphone is muted by the ContextPoint (Requirement **R1**). Furthermore, a *"meeting app"* and a *"presentation app"* are offered to the user (Requirement **R3**). The meeting app provides the user with meeting specific information that is preconfigured to show the goal and agenda of the meeting as well as all logged-in participants alongside with their shared profile information of this particular meeting. The presentation app lists all presentation files on the device as well as on the cloud storage associated to the user and offers the capability to start them in a slide show. Based on the information of the context model, the ContextPoint automatically deploys a Filter-Port that orders the presentations by their defined category, so that meeting related presentations are shown first (Requirement **R1**). For the slide show functionality each slide can be shown on a *presentation device* which is by default the screen of the device executing the app. Within the room a projector is installed which can be used remotely as a presentation device. Therefore, another notebook is connected to the projector via cable, running a SMAGs app that remotely offers the *"ISlideShowPresenter"* interface. The ContextPoint device offers to dynamically connect the projector with the presenter app to extend the display (Requirement **R2**). When the user agrees, the slide-show is automatically presented using the rooms projector. Whenever a slide changes, metadata about the slides is transferred along with the original content. The ContextPoint can dynamically include a Port Model within the presentation app that investigates the metadata of each slide when it is shown. When the brightness in the room is high

 ISBN: 978-1-61208-341-4

(sensed by the brightness sensor) and media content is shown (e.g., a video within the slide), the Port Model automatically controls the rooms light system to decrease the brightness and increase the visibility of the video (Requirements **R1** and **R2**). Afterwards it will illuminate the room again. This sample application was deployed on exemplary meeting room setup on a local exhibition. With the realization of this example we have shown that the presented approach supports the presented requirements within this scenario.

VI. Related Work

Much research has been done in the field of self-adaptive and context-aware systems. Especially, in the domain of mobile and ubiquitous computing, numerous research projects were conducted. One of the first context-aware systems was *ParcTab*, developed by Schilit in 1993 [16]. ParcTabs are individual mobile devices that are dynamically connected to other devices based on their location. At that time the major problem is to physically connect those devices using a heterogeneous network infrastructure. While those issues were solved over the last decades, current research problems mainly focus to autonomously provide the best suited services on the desired devices based on the users location, time, and surroundings.

Many context-aware and location-based applications have been developed. Bravo et al. presented a self-adaptive, context-aware conference application using RFID tags for localizing people within a conference building and distributed applications sensing a shared context (including the current location of the conference attendees) provided by a central server [17]. There are also other location-based services based on NFC-Tags for advertisement [18] and content delivery [18]. All these examples show that a lot of different use cases for context-aware systems exist, which all treat location as a central aspect. The commonality between these approaches is that they have implemented their own architecture designed for their specific, individual usage scenario. All those architectures support a subset of the adaptation capabilities of the ContextPoint approach. Thus, ContextPoint can be used as a platform for context-aware and location-based applications as it supports all required features of the discussed examples.

Another large research field concentrates on location-based services. Huebscher and McCann, for instance, presented a middleware for location-based, context-aware applications in smart home environments [19]. In their approach, the context (e.g., location, activity, etc.) is provided by *Context Services*, which analyse data delivered by one or more *Context Providers*. Based on the interpretation of the context, services are selected for a current activity of the user. This is similar to the Service Trader architecture of the ContextPoint approach. Nevertheless, they neglect that service selection is only one aspect of context-aware adaptation. Furthermore, traditional service-oriented approaches cannot individualize single services for multiple clients (only for every user or none). Since roles can adapt the behaviour of a player based on relationships, a single service instance can have different behaviour depending on the client using this service.

Other research projects aimed to design variable MAPE-K loops in order to adjust the adaptation process. In most of the cases, only the monitoring and analyse phase can be extended at runtime. The MUSIC project, for example, proposed a self-adaptive architecture for mobile devices with *Context Plug-Ins* [20]. This plug-in infrastructure enables to change the adaptation process. Even though, the MUSIC architecture does not prohibit the introduction and activation of plug-ins at runtime, it is only possible to extend the monitoring and analyse phase with new sensors and reasoners. In contrast, the ContextPoint architecture allows to exchange the operation of the whole MAPE-K loop (e.g., use alternative context model representations, introduce new planners or even change the control flow between the elements of the control loop).

Other approaches in the area of adaptive systems in mobile scenarios with context-aware extensible adaptation focus on the content presented on the device. While those approaches use a similar distributed architecture, they provide user-profile and device-capability adapted content [21]. The fine-grained structure of the applications cannot be adapted and the overall behaviour of an application cannot be changed. Van Sinderen et al. propose an architecture for context-aware adaptation for faster static evolution (i.e., at design-time) of self-adaptive systems [22]. Therefore, ECA-Rules are evaluated against a distributed context-management infrastructure to steer adaptation and context-dependent services. For adaptation they focus on component replacement and reconnection, the drawbacks are discussed in [6]. Like in the proposed ContextPoint architecture the monitoring and analyse phase can be distributed across the environment, while in their concept the plan and execute phase are integrated within the application. This, however, hinders the adaptation to unanticipated scenarios for mobile devices, because concepts that were not considered during design-time of the ECA rules cannot be handled at runtime. The presented Meta Adaptation architecture was first conceptualized by Perrouin et al. [23]. They describe how Meta Adaptation can be used to adapt MAPE-K loops at runtime in order to adjust the adaptation process to the requirements arising from contextual changes. The ContextPoint architecture can be seen as a concrete implementation of this concept. Combined with the reconfiguration capabilities of SMAGs, fine grained and cross-cutting reconfigurations of an applications MAPE-K loop can be modelled and realized. The proposed architecture for location-based external Meta Adaptation aims to dynamically connect local devices to build ad-hoc SoS. Weyns et al. proposed three different architectural styles of self-adaptation for SoS [24]. In his classification, the proposed Meta Adaptation architecture would be categorized as an instance of the *Collaborative Adaptations* architectural style with multiple hierarchical MAPE-K loops. These are able to include the information extracted in the monitoring phase and to reconfigure these loops during execution phase. The presented approach forms a Service-Oriented Architecture (SOA) [25], since all applications expose services that can be integrated into other applications. Traditionally, SOA-based approaches rely on Web Services. As discussed by Piechnick et al. [6], adaptation in classical Web-Service-based solutions use adaptive orchestration or choreography. On the one hand, the selection of services (i.e., which service instances), on the other hand, the process itself (i.e., control- and data-flow between the services) can be varied, to adapt the behaviour of the overall application. Especially the service selection corresponds to adaptation with component replacement (see Section II). In contrast, SMAGs allows for varying the behavior of a single

 ISBN: 978-1-61208-341-4

instance of a service based on the environmental situation and the calling instance without the need to replace/create entire service instances, which is important for stateful services, when the state cannot be transferred easily. Web Services can be used as a platform-independent communication infrastructure instead of the current socket-based realization in SMAGs, whereby the implementation of a service is a SMAGs component that can be adapted using roles.

VII. Conclusion and Future Work

Mobile devices changed the use and development of software fundamentally. In the future, users will expect that apps for mobile devices automatically adapt their behaviour based on their physical location, their user profile, and the current task. Furthermore, cheap, standardized, and easy to install sensors and actuators for smart environments offer new possibilities to gather environmental information. This in turn will extend the functionality of a mobile device towards environmental services. Traditionally, adaptive systems are based an a self-adaptive control loop within the application, which senses the environment and coordinates reconfiguration. In this paper, we showed how the adaptation architecture of Smart Application Grids can be used for Meta Adaptation and, in consequence, to support unanticipated scenarios. Because the MAPE-K loop of SMAGs applications is itself designed as SMAGs components, it can be adapted at runtime as well. This allows to create MAPE-K loops in other applications that reconfigure the adaptation process of the original self-adaptive system. Furthermore, we presented an architecture for smart environments, the ContextPoint approach, which aims to provide location-specific unanticipated adaptation. Therefore, the symbolic location of a mobile device is determined by NFC communication with a ContextPoint device. The ContextPoint offers location- and context-specific apps, a Meta Adaptation infrastructure to adapt the participating devices as well as applications running on them in unforeseen ways. Thus, it fully supports unanticipated adaptation, runtime application composition, and automatic application provisioning. For future work, security and privacy issues must be investigated, since those aspects are crucial for a real world application. Furthermore, it must be investigated if low cost computing devices are suitable to handle multiple participants. Additionally, it should be investigated, which of the architectural styles, according to Weyns et al. [24], are suitable for Meta Adaptation.

Acknowledgment

This work is supported by the German Research Foundation (DFG) within the Cluster of Excellence “Center for Advancing Electronics Dresden”, the Collaborative Research Center 912 “Highly Adaptive Energy-Efficient Computing” and the research training group ’Role-Based Software-Infrastructures’.

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] W. Pree, “Meta patterns - a means for capturing the essentials of reusable object-oriented design,” in Object-Oriented Programming. Springer, 1994, pp. 150–162.

[3] B. Morin et al., “An aspect-oriented and model-driven approach for managing dynamic variability,” in Model Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5301, pp. 782–796.

[4] OMG, Meta Object Facility (MOF) Core Specification Version 2.0, 2006. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01 [retrieved: April, 2014]

[5] U. Aßmann, Invasive software composition. Springer, 2003.

[6] C. Piechnick, S. Richly, S. Götz, C. Wilke, and U. Aßmann, “Using role-based composition to support unanticipated, dynamic adaptation-smart application grids,” in ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications, Nice, France, 2012, pp. 93–102.

[7] T. Reenskaug, P. Wold, and O. A. Lehne, Working with objects - the OOram software engineering method. Manning, 1996.

[8] J. Hillman and I. Warren, “Meta-adaptation in autonomic systems,” in Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of, 2004, pp. 292–298.

[9] A. K. Dey, “Understanding and using context,” Personal Ubiquitous Comput., vol. 5, no. 1, Jan. 2001, pp. 4–7.

[10] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications, ser. WMCSA ’94. Washington, DC, USA: IEEE Computer Society, 1994, pp. 85–90.

[11] S. Greenberg, “Context as a dynamic construct,” Hum.-Comput. Interact., vol. 16, no. 2, 2001, pp. 257–268.

[12] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, “Context is key,” Commun. ACM, vol. 48, no. 3, 2005, pp. 49–53.

[13] A. Zimmermann, A. Lorenz, and R. Oppermann, “An operational definition of context,” in Modeling and Using Context, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4635, pp. 558–571.

[14] A. Schmidt, M. Beigl, and H. w. Gellersen, “There is more to context than location,” Computers and Graphics, vol. 23, 1998, pp. 893–901.

[15] “Raspberry Pi,” http://www.raspberrypi.org/, visited 05/05/2014.

[16] B. Schilit, N. Adams, R. Gold, M. Tso, and R. Want, “The parctab mobile computing system,” in Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on, Napa, CA, 1993, pp. 34–39.

[17] J. Bravo, R. Hervs, I. Snchez, G. Chavira, and S. Nava, “Visualization services in a conference context: An approach by rfid technology,” j-jucs, vol. 12, no. 3, 2006, pp. 270–283.

[18] M. Hardt and S. Nath, “Privacy-aware personalization for mobile advertising,” in Proceedings of the 2012 ACM Conference on Computer and Communications Security, ser. CCS ’12. New York, NY, USA: ACM, 2012, pp. 662–673.

[19] M. C. Huebscher and J. A. McCann, “Adaptive middleware for context-aware applications in smart-homes,” in Proceedings of the 2Nd Workshop on Middleware for Pervasive and Ad-hoc Computing, ser. MPAC ’04. New York, NY, USA: ACM, 2004, pp. 111–116.

[20] N. Paspallis et al., “A pluggable and reconfigurable architecture for a context-aware enabling middleware system,” in On the Move to Meaningful Internet Systems, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5331, pp. 553–570.

[21] T. Lemlouma and N. Layaida, “Context-aware adaptation for mobile devices,” in Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on, Berkeley, CA, USA, 2004, pp. 106–111.

[22] M. van Sinderen, A. Van Halteren, M. Wegdam, H. Meeuwissen, and E. Eertink, “Supporting context-aware mobile applications: an infrastructure approach,” Communications Magazine, IEEE, vol. 44, no. 9, 2006, pp. 96–104.

[23] G. Perrouin et al., “Towards flexible evolution of dynamically adaptive systems,” in Proceedings of the 34th International Conference on Software Engineering, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1353–1356.

[24] D. Weyns and J. Andersson, “On the challenges of self-adaptation in systems of systems,” in Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems, ser. SESoS ’13. Montpellier, France: ACM, 2013, pp. 47–51.

[25] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA: Service Oriented Architecture Best Practices, 8th ed. Prentice Hall Professional Technical Reference, 2005.

 ISBN: 978-1-61208-341-4

Adaptive Scheduling of Smart Home Appliances Using Fuzzy Goal Programming

Honggang Bu and Kendall E. Nygard
Department of Computer Science
North Dakota State University
Fargo, ND, 58108, USA
Honggang.bu@my.ndsu.edu, kendall.nygard@ndsu.edu

***Abstract*-A smart electrical grid is highly instrumented and can be intelligently controlled. We describe a smart grid study in which we assume that utilities dynamically price electrical power to help regulate supply and demand balance, and that consumers have the ability to intelligently schedule times for the operation of their home appliances in response to prices. We present a mixed integer linear fuzzy goal programming with priorities imposed on different appliances. The goal programming formulation allows time preference constraints to be elastic rather than rigid. Another important feature of the model is flexibility of time slot delays for pairs of appliances for which the operation of one must follow the other (a washer/dryer pair for example). Numerical experimental results based on real spot prices for electricity are presented. In addition, computational time and the influence of time slot lengths and priorities are discussed.**

Keywords-Smart grid; Dynamic Pricing; Adaptive Systems; Optimization.

I. INTRODUCTION

Dynamic electricity pricing on an hourly basis is increasingly common in the United States [1]. This pricing policy is intended to help reduce system peak demand and also shift some load to off-peak, less expensive time periods. This can achieve more balance between energy demand and generation. Hourly pricing provides customers with opportunities to reduce their costs by managing the times at which electricity is consumed in the home. Smart appliances that can be accessed and controlled under the expanded addressing space of Internet Protocol version 6 (IPv6) are becoming common, and older appliances can be IP controlled through devices such as smart power bars.

Several studies on optimal scheduling of home appliances have been reported. Using Markov chains to model both energy prices and residential device usage, an energy management system called CAES for residential demand response applications to reduce residential energy costs and smooth energy usage was proposed [2]. In developing a Mixed Integer Linear Programming (MILP) problem formulation for electricity management in multiple homes, Oliveira et al. considered both cost and variations in the availability of the power supply [3]. Sou et al. proposed an MILP formulation with discrete time-slots [4]. In that model, one execution period (e.g., one day) is discretized into a prescribed number of uniform time slots. Amounts of energy are assigned to each time slot for each phase of appliance operation. Inspired by the model of Sou, Wu included the CO_2 footprint cost into the objective function by giving it a weight for modeling environmental concerns [5]. Giorgio developed a similar MILP formulation, but also included domestic renewable energy and batteries as energy sources [6].

In our work, we schedule home appliances using time slots and a MILP, based on portions of the existing work [4]. We expand the approach by adopting a fuzzy goal programming formulation [8]. Our model supports priority distinctions for the different appliances, and rigid time preference constraints are transformed into soft ones and included in the fuzzy goal programming framework with priorities. In addition, we devised constraints for modeling alternative delay times between running times of closely related appliances. The new method uses electricity prices known 24 hours in advance; so, the scheduling is exactly one day ahead.

Section II briefly introduces the concept of MILP and fuzzy goal programming; Section III presents the mathematical formulation for our mixed integer linear fuzzy goal programming model; Section IV provides the numerical experiments and results; and Section V presents the conclusions.

II. MILP AND GOAL PROGRAMMING

MILP is a widely used subset of mathematical programming in which the objective function is a linear function of the decision variables, which can be either integer or non-integer. Each constraint is formed from a linear combination of the decision variables [7].

When multiple, conflicting objectives or goals are involved in an optimization problem, goal programming is a powerful and effective tool. The two major differences of goal programming from conventional single-objective linear programming are the incorporation of flexibility in the constraint functions, and the satisficing approach that seeks a balanced and practical solution rather than an absolute optimal one [8]. To solve optimization problem with multiple conflicting goals using fuzzy goal programming, which is also called Chebyshev goal programming [8], each original single goal is first optimized to get the corresponding optimal goal value, then a solution that

 ISBN: 978-1-61208-341-4

minimizes the maximum deviation from any single optimized goal value is sought. Based on the degree of importance of each goal, priorities can be added to these deviations to reflect different penalties applied to different failures to meet the optimal goals [9]. A new general goal based on the weighted sum of deviations can therefore be formed and solved.

III. MATHEMATICAL FORMULATION

A. Assumptions and Parameters

We define an energy phase as an uninterruptible sub-process of the entire operation process of an appliance. Each appliance has a single phase or multiple energy phases that must be operated in sequence, with each using a pre-specified amount of electrical energy. The technical specifications of appliances defined by the manufacturers of appliances must be met. Constraints are employed to ensure the sequential operations of some appliances, to model the delay between the running of two closely related appliances, to ensure that the total energy consumed within a certain period does not exceed the peak energy allowed, and to ensure that user time preferences are met. The overall objective of the model is to produce the schedule for running the appliances that saves a consumer as much energy cost as possible, while meeting all of the constraints. Our MILP formulation is for a single 24-hour day. Each hour is uniformly discretized into h time slots, so that the number of total time slots in a day is $m = 24^* h$. N is the number of appliances, and for each appliance i ($i = 1,2,\dots,N$), $n_i$is the number of uninterruptible energy phases for each appliance.

Parameters $\lambda_g \left(0 < \lambda_g < 1, and\ g = 1,2,\dots,N+1\right)$, satisfying $\sum_{g=1}^{N+1} \lambda_g = 1$, are used to model the priorities assigned to each single deviation goal in the fuzzy goal programming model. Here, $\lambda_g (g = 1,2,\dots,N)$ is for the deviation goals for each corresponding appliance energy cost, and λ_{N+1} is the priority for the user time preference penalty deviation goal. $\lambda_g (0 < \lambda_g < 1, g = 1,2,\dots,N+1)$ is specified by the user according to their preferences for different appliances.

T_{ij} $(i = 1,2,\dots,N, j = 1,2,\dots,n_i)$represents the nominal processing time for energy phase j for appliance i in minutes, $\underline{\gamma}$ and $\overline{\gamma}$ $(0.5 < \underline{\gamma} < 1 < \overline{\gamma} < 1.5)$ are lower and upper processing time limit factors for energy phase j of appliance i. To denote the lower and upper limits of power assignment, respectively, to the corresponding energy phase, $\underline{P}_{ij}^{k}$ and $\overline{P}_{ij}^{k}$ are introduced. The delay between two energy phases of an appliance is restricted by $\underline{D}_{ij}$ and $\overline{D}_{ij}$, the appliance technical specifications defining the lower and upper delay time, respectively, in minutes. E_{ij} is used to denote the total energy that a phase should use according to the technical specification.

B. Decision variables

Real (continuous) decision variables p_{ij}^{k} $(k = 1,2,\dots,m; i = 1,2,\dots,N; j = 1,2,..,n_i)$ are used to indicate the energy assigned to energy phase j of appliance i during the period of time slot k.

To indicate during time slot k whether a particular energy phase j of appliance i is being processed, a series of binary decision variables $x_{ij}^{k} \in \{0,1\}$ are used, with $x_{ij}^{k} = 1$ indicating energy phase being processed, and otherwise not being processed.

Binary variables $s_{ij}^{k} \in \{0,1\}$ are utilized to indicate whether the processing of a particular energy phase is already finished by a particular time slot. If and only if $s_{ij}^{k} = 1$, energy phase j of appliance i is complete by time slot k.

To indicate whether appliance i is making a transition between energy phase $j-1$ to j at time slot k, binary variables t_{ij}^{k} $(j = 2,\dots,n_i)$ are utilized. $t_{ij}^{k} = 1$ if and only if during time slot k, the appliance i has finished energy phase $j-1$ in some earlier time slot, but the energy phase j has not yet started. These variables are useful for restricting the delay between energy phases of an appliance.

For the fuzzy goals, parameters $\delta_g (\delta_g > 0, g = 1,2,\dots,N+1)$ are introduced to denote the normalized maximum deviation between the best and the worst values of each single objective function. Specifically, $\delta_g (g = 1,2,\dots,N)$ are for the corresponding appliances, and δ_{N+1} is for the user time preference.

C. Constraints

1) Single appliance energy cost objective function: The total electricity cost for appliance i during the entire execution period, denoted by $Z_i (i = 1,2,\dots,N)$, is

$$\sum_{k=1}^{m} \sum_{j=1}^{n_i} c^k p_{ij}^{k} \quad (1)$$

where, c^k denotes the electricity price for time slot k.

2) Objective function for user time preference violation penalty: Here, we consider a simple user time preference in which the household user divides the day into two general parts: one that can be used to run a certain appliance and the other one cannot. Rather than use rigid constraints to absolutely prohibit using an appliance during the non-preferred time, we allow the time period to be used but impose a penalty on doing so. Let $TP_i^k \in \{0,1\}$ denote the user time preference interval, and $TP_i^k = 0$ if and only if none of the energy phase of appliance i is to be run during time slot k. Assume k_{start}^{i}, k_{mid}^{i}, and k_{end}^{i} is the first, middle, and the last slot number of the whole user prohibited time period (which is continuous) for appliance i, respectively; then the penalty for using prohibited time is expressed as

$$\sum_{i=1}^{N} \sum_{j=1}^{n_i} \sum_{k=k_{start}^{i}}^{k_{end}^{i}} x_{ij}^{k} \alpha^{-\left|k-k_{mid}^{i}\right|} \quad (2)$$

where, $\alpha > 1$ is a constant, and $k_{mid}^{i} = \left[(k_{start}^{i} + k_{end}^{i})/2\right]$. This is the objective function for the violation penalty for a user time preference, and is denoted as Z_{N+1}. Note that this function is a weighted penalty in that the closer to the middle of the prohibited time zone, the higher penalty that results.

3) Maximum single objective deviation constraints: Let U_i and L_i be the best possible and worst possible values, respectively, for the k^{th} single objective, then we have the following constraints:

$$(\beta U_i - Z_i)/(\beta U_i - L_i) \le \delta_i, i = 1,2,\dots,N+1 \quad (3)$$

$$\delta_i \ge 0 \in \mathbb{R}, i = 1,2,\dots,N+1 \quad (4)$$

Each $\delta_i (i = 1,2,\dots,N+1)$ represents the worst deviation level for the k^{th} objective. Each U_i and L_i are obtained by optimizing corresponding Z_i and $-Z_i$ alone, respectively, without regard to other objectives. The expression $(\beta U_i - L_i)$ in (3) helps normalize the objective deviation level and thus adjust different levels to similar fluctuation ranges. With the normalized deviation levels, applying desired priorities to different objectives is easier. In consideration of the interaction between or among appliances and user time preferences, U_i may be close to but not the real possible optimal single objective value. So, an auxiliary coefficient β is incorporated to U_i to help use a better objective value than the "false" best possible value. Since in this study the best single objective value is the minimum value, β should be a positive constant and less than 1.

4) Sequential processing between appliances: Suppose appliance $\tilde{\imath}$ must be finished before appliance i starts (for example, the washing machine operations must be finished before the dryer starts), then the following constraint restricting the relationship between the last energy phase of the appliance $\tilde{\imath}$ and the first energy phase of appliance i must be satisfied:

$$s_{in_{\tilde{\imath}}}^{k} \ge x_{i1}^{k}, \forall k \quad (5)$$

5) Between-appliance delay: In reality, some appliances are more closely related than just following the constraints restricting their sequential processing. For example, the dryer can start running only after the washing machine is finished, as specified by (5), and in practice the delay between the two appliances usually cannot be very large. Suppose, for example, that if the dryer must start working within 3 time slots after the washing machine is done, then the following constraints holds:

$$s_{in_{\tilde{\imath}}}^{k} - s_{in_{\tilde{\imath}}}^{k-1} - x_{in_{\tilde{\imath}}}^{k} \le x_{i1}^{k} + x_{i1}^{k+1} + x_{i1}^{k+2}$$
$$\forall k = 2,3,\dots,m-2, \quad (6)$$

These constraints should be used together with (5), namely, appliance $\tilde{\imath}$ and i must satisfy (5) first.

To establish that these constraints are theoretically correct, consider the logic below.

If k_0 is the first time slot after the last energy phase of appliance $\tilde{\imath}$ is finished, then $k_0 - 1$ is the last slot when the last energy phase of appliance $\tilde{\imath}$ is being processed. This also implies:

i) When $k < k_0$, $s_{in_{\tilde{\imath}}}^{k} = s_{in_{\tilde{\imath}}}^{k-1} = 0$, and $x_{in_{\tilde{\imath}}}^{k} = 0$ *or* 1, so, the left side of the constraints is always equal to or less than 0. In this case, the constraints hold. Also in this case, the appliances sequential processing constraints ensure that all $x_{i1}^{k} = 0, \forall k < k_0$.

ii) When $k = k_0$, $s_{in_{\tilde{\imath}}}^{k} = 1, s_{in_{\tilde{\imath}}}^{k-1} = 0$, and $x_{in_{\tilde{\imath}}}^{k} = 0$, so, the left side of the constraints is always equal to 1. In this case, the constraints require that at least one of the time slots right after the finishing of the previous appliance must be used to start processing of the second appliance.

iii) When $k > k_0$, $s_{in_{\tilde{\imath}}}^{k} = s_{in_{\tilde{\imath}}}^{k-1} = 1$, and $x_{in_{\tilde{\imath}}}^{k} = 0$, so, the left side of the constraints is always equal to 0. In this case, the constraints hold.

6) Sequential processing between energy phases: Usually an energy phase of an appliance cannot start working unless its preceding phases have finished. The following constraints specify this condition:

$$s_{i(j-1)}^{k} \ge x_{ij}^{k}, \forall i, k, \forall j = 2,3,\dots,n_i \quad (7)$$

7) Between-phase delay: The delay between two energy phases of an appliance is restricted to a specific range. Suppose that $\underline{D}_{ij}$ and $\overline{D}_{ij}$ are the appliance technical specifications defining the lower and upper delay, respectively, in minutes, then the following constraints must be satisfied:

$$\left\lceil \frac{\underline{D}_{ij}}{60} h \right\rceil \le \sum_{k=1}^{m} t_{ij}^{k} \le \left\lfloor \frac{\overline{D}_{ij}}{60} h \right\rfloor, \forall i, \forall j = 2,3,\dots,n_i \quad (8)$$

$$t_{ij}^{k} = s_{i(j-1)}^{k} - x_{ij}^{k} - s_{ij}^{k}, \forall i, k, \forall j = 2,3,\dots,n_i \quad (9)$$

8) Uninterruptible operation of an energy phase: To ensure the integrity and continuity of an energy phase, the following constraints should be satisfied:

$$x_{ij}^{k} \le 1 - s_{ij}^{k}, \forall i, j, k \quad (10)$$

$$x_{ij}^{k-1} - x_{ij}^{k} \le s_{ij}^{k}, \forall i, j, \forall k = 2,3,\dots,m \quad (11)$$

$$s_{ij}^{k-1} \le s_{ij}^{k}, \forall i, j, \forall k = 2,3,\dots,m \quad (12)$$

9) Energy phase process time limits: Process time limits are enforced by the following constraint:

$$\left\lceil \frac{T_{ij}h}{60} \underline{\gamma} \right\rceil \le \sum_{k=1}^{m} x_{ij}^{k} \le \left\lfloor \frac{T_{ij}h}{60} \overline{\gamma} \right\rfloor, \forall i, j \quad (13)$$

where h is the number of time slots in each hour, T_{ij} is the nominal processing time for energy phase j in appliance i in minutes, $\underline{\gamma}$ and $\overline{\gamma}$ $(0.5 < \underline{\gamma} \le 1 \le \overline{\gamma} < 1.5)$ are the lower and upper processing time limits factor for energy phase j in appliance i.

10) Energy phase energy assignment requirement and bounds: Each energy phase uses a certain amount of energy E_{ij} specified by the manufacturer:

$$\sum_{k=1}^{m} p_{ij}^{k} = E_{ij}, \forall i, j \quad (14)$$

To ensure power safety, the total energy assigned in any time slot is not allowed to exceed the peak signal or in other words the total slot energy upper bound:

$$\sum_{i=1}^{N} \sum_{j=1}^{n_i} p_{ij}^{k} \le PEAK^{k}, \forall \mathrm{k} \quad (15)$$

The energy assignment in any time slot for each energy phase of each appliance should satisfy the following constraint:

 ISBN: 978-1-61208-341-4

$$\frac{\underline{P}_{ij}^k}{h} x_{ij}^k \le p_{ij}^k \le \frac{\overline{P}_{ij}^k}{h} x_{ij}^k, \forall i,j,k \quad (16)$$

where, $\underline{P}_{ij}^k$ and $\overline{P}_{ij}^k$ are the lower and upper limits of power (not energy) assignment, respectively, to the corresponding energy phase. These limits are specified by the appliance manufacturer.

11) Basic decision variable constraints:

$$p_{ij}^k \ge 0 \in \mathbb{R}, \forall i,j,k \quad (17)$$

$$x_{ij}^k \in \{0,1\}, \forall i,j,k \quad (18)$$

$$s_{ij}^k \in \{0,1\}, \forall i,j,k \quad (19)$$

$$t_{ij}^k \in \{0,1\}, \forall i,k \; \forall j = 2,\dots,n_i \quad (20)$$

D. Cost function

Finally, the following total cost function, which represents the weighted sum of the maximum objective deviation from each single goal, is specified:

$$\sum_{i=1}^{N+1} \lambda_i \delta_i \quad (21)$$

E. General formulation

The general formulation of the proposed framework is summarized as follows:

$$\text{minimize}_{p,x,s,t,\delta} \quad \text{Cost function (21)}$$

$$\text{Subject to:} \quad \text{Constraints (3)-(20)}$$

This is a MILP formulation transformed from the fuzzy goal programming formulation, and it can be solved using classical algorithms or heuristic search methods [4][5].

IV. NUMERICAL EXPERIMENTS

All experiments were conducted on a desktop computer with an Intel^{R} Core^{TM} 3.40GHz CPU and 16GB RAM. The optimization problem was solved using MATLAB interface of YALMIP and IBM ILOG CPLEX 12.5 solver [10][11][12].

The 24-hour ahead hourly electricity price data of Nov. 3rd, 2013, for Long Island of New York State used in this paper was taken from the NYISO [13]. From midnight to next midnight, these predicted pricing data in USD/MWh were 32.19, 27.63, 26.51, 24.6, 26.41, 22.57, 27.21, 28.6, 31.45, 35.64, 36.35, 36.86, 36.87, 36.21, 34.82, 35.17, 41.37, 57.86, 54.65, 55.44, 50.31, 45.73, 39.02, and 35.67. From these data it can be seen that the highest price (57.86) was 2.56 times the lowest price (22.57).

This study involved three controllable same smart home appliances including a dish washer, a washing machine, and a dryer, similar to those used by Sou et al. [4]. Three different lengths of time slot, 3 minutes, 5 minutes, and 10 minutes, were investigated in the numerical experiments. The dishwasher is not supposed to be run during midnight to 7 o'clock in the morning, and both the washing machine and dryer are not supposed to be run during midnight to 6 o'clock in the morning. The parameter α, which is the penalty term for using user prohibited time, was set to 1.1. The dryer can only start working after the washing machine has finished, and the delay between them should be no more than 3 time slots. The parameters $\underline{D}_{ij}$ in (8) for all phases is assumed to be 0, and $\overline{D}_{ij}$ in (8) for the dishwasher, washing machine and dryer were set to 5, 10, and 0 minutes, respectively. The parameters $\underline{\gamma}$ and $\overline{\gamma}$ in (13) were set to 0.8 and 1.2, respectively. The peak signals in (15) for 3-minute, 5-minute, and 10-minute time slots are assumed to be 3300 Wh, 5500 Wh, and 11000 Wh, respectively. The dishwasher, washing machine and dryer have 6, 8, and 1 energy phases, respectively. The parameter β in (3) was set to 0.5. The detailed technical specifications of the three appliances are shown in Table I through Table III. All of the rest of the parameter values can be found in these tables. Three representative user priority combinations $\lambda_g (g = 1,2,3,4)$ for the objective function were selected for study, as is listed in Table IV. In reality, all the priority choices are made by the users and completely up to them with regard to their preferences.

TABLE I. DISHWASHER TECHNICAL SPECIFICATIONS

energy phase	Energy required (Wh)	Min power (W)	Max power (W)	Nominal operation time (min)
pre-wash	16	6.47	140	14.9
Wash	751.2	140.26	2117.8	32.1
1st rinse	17.3	10.28	132.4	10.1
Drain	1.6	2.26	136.2	4.3
2nd rinse	572.3	187.3	2143	18.3
drain & dry	1.7	0.2	2.3	52.4

TABLE II. WASHING MACHINE TECHNICAL SPECIFICATIONS

energy phase	Energy required (Wh)	Min power (W)	Max power (W)	Nominal operation time (min)
movement	118	27.231	2100	26
pre-heating	5.5	5	300	6.6
Heating	2054.9	206.523	2200	59.7
Maintenance	36.6	11.035	200	19.9
Cooling	18	10.8	500	10
1st rinse	18	10.385	700	10.4
2nd rinse	17	9.903	700	10.3
3rd rinse	78	23.636	1170	19.8

TABLE III. DRYER TECHNICAL SPECIFICATIONS

energy phase	Energy required (Wh)	Min power (W)	Max power (W)	Nominal operation time (min)
Drying	2426.3	120.51	1454	120.8

 ISBN: 978-1-61208-341-4

TABLE IV. USER PRIORITIES FOR THREE APPLIANCES AND USER TIME PREFERENCE

Priority choice	**1**	**2**	**3**
Dishwasher	0.2	0.1	0.4
washing machine	0.2	0.2	0.3
Dryer	0.3	0.2	0.2
user time preference	0.3	0.5	0.1

V. DISCUSSION OF RESULTS

A. Computational time

Prematurely terminating the optimization process using the first feasible solution terminating condition can dramatically save computational time and at the same time have little influence on the final objective function value. Table V lists the relative extra time cost in using the default optimal solution terminating condition compared to using the first feasible solution terminating condition. Table VI shows the relative objective function error between the two terminating strategies. In view of this fact, our study adopted the first feasible terminating strategy in the remaining experiments.

TABLE V. RELATIVE EXTRA TIME COST (%)

Priority choice	**10-min time slot**	**5-min time slot**	**3-min time slot**
1	121.9013	290.0044	171.1622
2	49.74624	193.8439	158.5955
3	19.67994	225.2939	328.6174
average	63.77581	236.3807	219.4584

TABLE VI. RELATIVE OBJECTIVE ERROR (%)

Priority choice	**10-min time slot**	**5-min time slot**	**3-min time slot**
1	4.107487	3.16173	2.773725
2	2.172829	3.354763	4.889764
3	1.636755	1.867869	0.46734
average	2.639024	2.794787	2.710276

B. Influence of time slot length on electricity cost

The relative extra total electricity cost using the worst solution instead of the best solution is used to facilitate the investigation of the influence of time slot length on electricity cost, and the results are shown in Table VII. Here, the total electricity cost refers to the sum of the three single appliance energy cost objectives specified in (1). From Table VII, it can be seen that the worst-case total energy cost is approximately double that of the best case. The average relative extra energy cost for each time slot indicates no obvious cost savings between the 10-min time slot and the 5-min time slot, while the 3-min time slot can save significant money. This is because the smaller the time slot length, the more flexibility for appliances scheduling. However, the computation time for the 3-min time slot case is more than 3 times and 10 times that for the 5-min and 10-min time slot case, respectively. This illustrates the tradeoff between time slot size and computational time.

TABLE VII. RELATIVE EXTRA TOTAL ELECTRICITY COST (%)

Priority choice	**10-min time slot**	**5-min time slot**	**3-min time slot**	**average for each priority**
1	98.25	103.91	112.22	104.79
2	101.40	102.01	110.13	104.52
3	113.56	106.73	113.64	111.31
average for each time slot	104.40	104.22	112.00	106.87 (overall average)

C. Influence of objective priority choice on electricity cost

The influence of the single objective priority choice made by the user can be seen from Table VII. Priority choice 3 (0.4, 0.3, 0.2, 0.1) saves more money because of its very low user time preference priority and hence more prohibited time used. Using Choice 1 (0.2, 0.2, 0.3, 0.3) and Choice 2 (0.1, 0.2, 0.2, 0.5), similar results were produced. There are three reasons for this. First, the user time preference deviation objective δ_4 has great influence on the general fuzzy goal objective relative to appliance-related deviation objectives. We observe that the average optimal value of the user time preference-related deviation objective is more than 10 times that any of the other single objective values. Although the use of the worst and best objective values in constructing the fuzzy objective dramatically resizes the fluctuation range to a range that is similar to that of the other objectives, the great difference still exists. Second, the user time preference objective priority level 0.3 and 0.5 have almost the same effects on this single objective value. Third, only three appliances were involved in this study and two of them are closely related, resulting in small scheduling flexibility. Separately selecting and treating the user time preference objective priority levels can produce better effects.

Illustrations of the price data (Fig. 1) used in this study and some typical energy assignment examples (Fig. 2 – Fig. 4) are given below. All the energy assignment examples are based on the 10-min time slot. With the increase of the user time preference-related objective priority, the violation of the user prohibited time decreases, and eventually no prohibited time is used when this priority is very high.

 ISBN: 978-1-61208-341-4

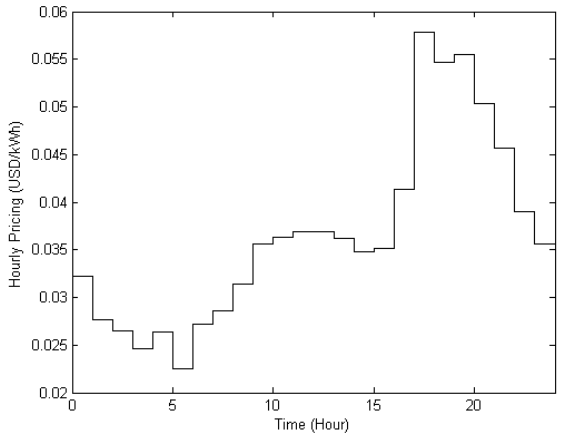


Figure 1. Hourly pricing data.

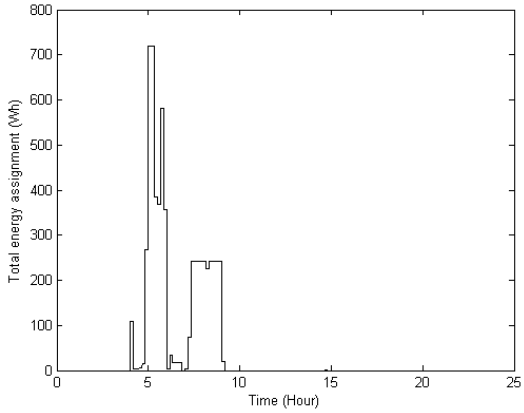


Figure 2. Total energy assignment under priority choice 3 (0.4, 0.3, 0.2, 0.1).

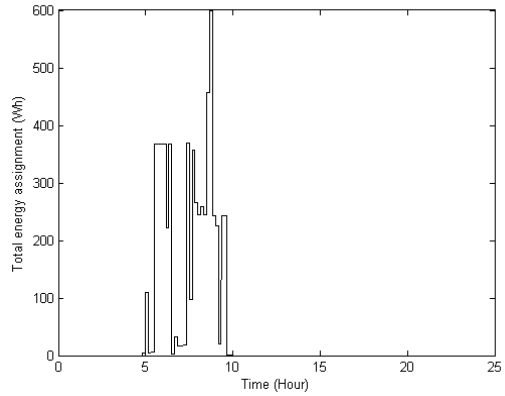


Figure 3. Total energy assignment under the priority choice (0.14, 0.23, 0.20, 0.43).

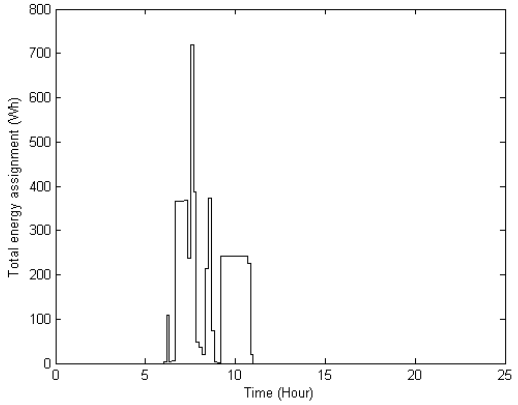


Figure 4. Total energy assignment under the priority choice (0.02, 0.04, 0.04, 0.9).

D. Comparative discussion

The research of applying adaptive fuzzy goal programming theory with priority considered to the area of optimal scheduling of smart home appliances is still quite new. Compared with the reported existing models, our model is more realistic and practical with the new developed between-appliance delay constraints, the soft user time preferences, and the priorities imposed on each single objective. In [4] and [14], only the plain MILP formulation was used to model the home appliance scheduling problem based on rigid user time preferences and without considering the priority of each appliance. A very simple household appliances scheduling formulation taking into account only the peak hourly load constraints was proposed [15]. Samadi et al. [16] proposed a real-time residential load scheduling that took consideration of the load uncertainty, but quite different energy phase concepts such as sleep, awake, active, finished, etc., were used. A type of semi-soft user time preference constraints were proposed in home appliances scheduling [17], however, there are two limitations in this study: the user time preference constraint under each discrete sensitivity level was still a rigid one; no energy phase concepts were adopted for detailed investigation. Direct comparison of the energy saving between the proposed model and other models would be totally meaningless as each model was established based on quite different assumptions, objective, and constraints. In many cases, electricity cost is only part of the general objective.

The proposed fuzzy goal programming model for home appliances scheduling does involve more variables and constraints than does a plain MILP model. The implementation time required for our model is 13% more on average than that for [3]. Since the proposed scheduling is supposed to make one day in advance and only a few minutes or even less than one minute is needed to finish the implementation, the extra time cost becomes marginal.

 ISBN: 978-1-61208-341-4

VI. CONCLUSION AND FUTURE WORK

The proposed mixed integer fuzzy goal programming model for adaptive scheduling of smart home appliances was shown to be effective in saving user's total electricity cost. The user time preferences were transformed from rigid constraints to soft violation penalty objectives and integrated into the fuzzy goal programming formulation. Our optimization solution also allows users to give preferred priorities to different appliances objectives and the user time preference objective as well. The newly introduced constraints that restrict the delay between two closely related appliances make the proposed framework practical. More appliances with same or different type are to be included in the future research to further investigate the performance of the proposed method. The general conclusion of the study is that a closed-form optimization model is an effective approach for adaptation of home appliance schedules to changing prices of electrical power. Future work is to include more common and frequently-used smart home appliances in the study to further test the validity of the proposed model.

REFERENCES

[1] F. A. Wolak, "Do Residential Customers Respond to Hourly Prices? Evidence from a Dynamic Pricing Experiment.", American Economic Review: Papers & Proceedings, vol. 101:3, pp. 83–87.

[2] D. O'Neill, M. Levorato, A. Goldsmith, and U. Mitra, "Residential Demand Response Using Reinforcement Learning", IEEE SmartGrid-Comm., Sep. 2010, Stanford University report.

[3] G. D. Oliveira, M. Jacomino, D. L. Ha, and S. Ploix, "Optimal Power Control for Smart Homes," 18th IFAC World Congress, Aug. – Sep. 2011, pp. 9579-9586, ISSN: 1474-6670, ISBN: 978-3-902661-93-7.

[4] K. C. Sou, J. Weimer, H. Sandberg, and K. H. Johansson, "Scheduling Smart Home Appliances Using Mixed Integer Linear Programming," 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), IEEE Press, Dec. 2011, pp. 5144-5149, doi: 10.1109/CDC. 2011.6161081.

[5] J. Wu, "Scheduling smart home appliances in the Stockholm Royal Seaport", Degree project in Automatic Control Master's Degree Project, Aug. 2012, Stockholm, Sweden.

[6] S. Giorgio, "Optimal Scheduling of Smart Home Appliances Using Mixed-Integer Linear Programming," Master thesis of Università degli Studi di Padova. Dec. 2012.

[7] J. C. Smith and Z. C. Taskin, "Appendix A: Tutorial Guide to Mixed Integer Programming Models and Solution Techniques," in G. J. Lim and E. K. Lee, Editors, Optimization in Medicine and Biology, Auerbach Publications, 2007, eBook ISBN: 978-0-8493-0569-6, doi: 10.1201/9780849305696.axa, pp.522-546.

[8] J. P. Ignizio and C. Romero, "Goal Programming", in H. Bidgoli, Editor, Encyclopedia of Information Systems, Vol. 2, 2003, Academic Press, pp. 489-500.

[9] C. F. Hu, C. J. Teng, and S. Y. Li, "A Fuzzy Goal Programming Approach to Multi-Objective Optimization Problem with Priorities," European Journal of Operational Research, vol. 176, 2007, pp. 1319-1333, doi: 10.1016/j.ejo r .2005 .10.049.

[10] The Mathworks, Inc., MATLAB 8.0. Natick, MA, USA, 2012.

[11] J. Löfberg, "YALMIP : A Toolbox for Modeling and Optimization in MATLAB," Proceedings of the CACSD Conference, Sep. 2004, pp. 284-289, ISBN: 0-7803-8636-1, doi: 10.1109/CACSD.2004.1393890.

[12] IBM, IBM ILOG CPLEX 12.5. Armonk, New York, 2013.

[13] NYISO, "Pricing Data", http://www.nyiso.com/ public /marke ts_operations/market_data/pricing_data/index.jsp, 2013.

[14] A. Agnetis, G. Dellino, P. Detti, G. Innocenti, G. D. Pascale, A. Vicino, "Appliance Operation Scheduling for Electricity Consumption Optimization", 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), IEEE Press, Dec. 2011, pp. 5899-5904, doi: 10.1109/CDC.2011.6160450.

[15] T. Yu, D. S. Kim, and S. Y. Son, "Home Appliance Scheduling Optimization with Time-Varying Electricity Price and Peak Load Limitation", Proceedings of 2nd International Conference on Information Science and Technology, ASTL vol. 23, 2013, pp. 196 – 199.

[16] P. Samadi, H. Mohsenian-Rad, V.W.S. Wong, and R. Schober, "Tackling the Load Uncertainty Challenges for Energy Consumption Scheduling in Smart Grid", IEEE Transactions on Smart Grid, vol. 4, 2013, pp. 1007-1016, doi: 10.1109/TSG.2012.2234769.

[17] B. Saha, "Scheduling of Appliances based on Sensitivity to Dynamic Pricing in a Smart Grid", Master Thesis, Fargo, ND, USA, Sep. 2013.

 ISBN: 978-1-61208-341-4