DEPEND 2014

The Seventh International Conference on Dependability

ISBN: 978-1-61208-378-0

November 16 - 20, 2014

Lisbon, Portugal

DEPEND 2014 Editors

Rolf Johansson, SP, Sweden

Carla Westphall, Federal University of Santa Catarina, Brazil

DEPEND 2014

Foreword

The Seventh International Conference on Dependability (DEPEND 2014), held between November 16-20, 2014 in Lisbon, Portugal, provided a forum for detailed exchange of ideas, techniques, and experiences with the goal of understanding the academia and the industry trends related to the new challenges in dependability on critical and complex information systems.

Most of critical activities in the areas of communications (telephone, Internet), energy & fluids (electricity, gas, water), transportation (railways, airlines, road), life related (health, emergency response, and security), manufacturing (chips, computers, cars) or financial (credit cards, on-line transactions), or refinery& chemical systems rely on networked communication and information systems. Moreover, there are other dedicated systems for data mining, recommenders, sensing, conflict detection, intrusion detection, or maintenance that are complementary to and interact with the former ones.

With large scale and complex systems, their parts expose different static and dynamic features that interact with each others; some systems are more stable than others, some are more scalable, while others exhibit accurate feedback loops, or are more reliable or fault-tolerant.

Inter-system dependability and intra-system feature dependability require more attention from both theoretical and practical aspects, such as a more formal specification of operational and non-operational requirements, specification of synchronization mechanisms, or dependency exception handing. Considering system and feature dependability becomes crucial for data protection and recoverability when implementing mission critical applications and services.

Static and dynamic dependability, time-oriented, or timeless dependability, dependability perimeter, dependability models, stability and convergence on dependable features and systems, and dependability control and self-management are some of the key topics requiring special treatment. Platforms and tools supporting the dependability requirements are needed.

As a particular case, design, development, and validation of tools for incident detection and decision support became crucial for security and dependability in complex systems. It is challenging how these tools could span different time scales and provide solutions for survivability that range from immediate reaction to global and smooth reconfiguration through policy based management for an improved resilience. Enhancement of the self-healing properties of critical infrastructures by planning, designing and simulating of optimized architectures tested against several realistic scenarios is also aimed.

To deal with dependability, sound methodologies, platforms, and tools are needed to allow system adaptability. The balance dependability/adaptability may determine the life scale of a complex system and settle the right monitoring and control mechanisms. Particular challenging issues pertaining to context-aware, security, mobility, and ubiquity require

appropriate mechanisms, methodologies, formalisms, platforms, and tools to support adaptability.

We take here the opportunity to warmly thank all the members of the DEPEND 2014 Technical Program Committee, as well as the numerous 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 DEPEND 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 DEPEND 2014 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that DEPEND 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 field of dependability.

We are convinced that the participants found the event useful and communications very open. We hope Lisbon provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

DEPEND 2014 Chairs:

DEPEND Advisory Chairs
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

DEPEND 2014 Industry Liaison Chairs
Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2014 Research/Industry Chair
Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2014 Special Area Chairs
Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan
Hardware dependability
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy
Security and Trust
Syed Naqvi, CETIC, Belgium

DEPEND 2014

Committee

DEPEND Advisory Chairs

Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

DEPEND 2014 Industry Liaison Chairs

Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2014 Research/Industry Chair

Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2014 Special Area Chairs

Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan

Hardware dependability
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany

Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy

Security and Trust
Syed Naqvi, CETIC, Belgium

DEPEND 2014 Technical Program Committee

Don Adjeroh, West Virginia University, USA
Muhammad Afzaal, National University of Computer and Emerging Sciences, Pakistan
Jose Ignacio Aizpurua Unanue, University of Mondragon, Spain
Murali Annavaram, University of Southern California, USA
Afonso Araújo Neto, University of Coimbra, Portugal
José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain
Radu F. Babiceanu, Embry-Riddle Aeronautical University, USA
Ian Bayley, Oxford Brookes University, U.K.
Siegfried Benkner, University of Vienna, Austria
Jorge Bernal Bernabé, University of Murcia, Spain
James Brandt, Sandia National Laboratories, U.S.A.

Ezzat Kirmani, St. Cloud State University, USA
Seah Boon Keong, MIMOS Berhad, Malaysia
Abdelmajid Khelil, Huawei Research, Germany
Kenji Kono, Keio University, Japan
Israel Koren, University of Massachusetts - Amherst, USA
Mani Krishna, University of Massachusetts - Amherst, USA
Mikel Larrea, University of the Basque Country UPV/EHU, Spain
Inhwan Lee, Hanyang University - Seoul, Korea
Matthew Leeke, University of Warwick, UK
Jane W. S. Liu, Academia Sinica, Taiwan
Yun Liu, Boeing Company, USA
Paolo Lollini, Dipartimento di Matematica e Informatica "U. Dini", Italy
Xuanwen Luo, Sandvik Mining, USA
Miroslaw Malek, Humboldt-Universitaet zu Berlin, Germany
Amel Mammar, Mines Telecom/ Telecom SudParis, France
Antonio Mana Gomez, University of Malaga, Spain
Gregorio Martinez, University of Murcia, Spain
Rivalino Matias Jr., Federal University of Uberlandia, Brazil
Yutaka Matsuno, Nagoya University, Japan
Manuel Mazzara, Polytechnic of Milan, Italy
Per Håkon Meland, SINTEF ICT, Norway
Carlos Julian Menezes Araujo, Federal University of Pernambuco, Brazil
Francesc D. Muñoz-Escoí, Universitat Politècnica de València, Spain
Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong
Jun Na, Northeastern University, China
Syed Naqvi, CETIC, Belgium
Sarmistha Neogy, Jadavpur University, India
Mats Neovius, Åbo Akademi University - Turku, Finland
Hong Ong, MIMOS Bhd, Malaysia
Frank Ortmeier, Otto-von-Guericke-Universität Magdeburg, Germany
Roberto Palmieri, Virginia Tech, USA
Aljosa Pasic, ATOS Origin, Spain
Ronald Petrlic, Saarland University, Germany
Alfredo Pironti, INRIA Paris Rocquencourt, France
Wolfgang Pree, University of Salzburg, Austria
Feng Qin, Ohio State University, USA
Rolf Riesen, IBM Research, Ireland
Ruben Rios, University of Málaga, Spain
Paolo Romano, INESC-ID/IST, Portugal
Christian Rossow, VU University Amsterdam, Netherlands
Francesca Saglietti, University of Erlangen-Nuremberg, Germany
Felix Salfner, SAP Innovation Center - Potsdam, Germany
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sahra Sedighsarvestani, Missouri University of Science and Technology, U.S.A.
Jean-Pierre Seifert, Technische Universität Berlin & Telekom Innovation Laboratories, Germany
Dimitrios Serpanos, University of Patras & ISI, Greece
Muhammad Shafique, Karlsruhe Institute of Technology (KIT), Germany
Kuei-Ping Shih, Tamkang University, Taiwan

Francois Siewe, De Montfort University, UK
Navjot Singh, Avaya Labs Research, USA
Alessandro Sorniotti, IBM research - Zurich, Switzerland
George Spanoudakis, City University London, U.K.
Kuo-Feng Ssu, National Cheng Kung University, Taiwan
Vladimir Stantchev, Berlin Institute of Technology, Germany
Neeraj Suri, TU-Darmstadt, Germany
Kenji Taguchi, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Oliver Theel, University Oldenburg, Germany Sergio Pozo Hidalgo, University of Seville, Spain
Kishor Trivedi, Duke University - Durham, USA
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Elena Troubitsyna, Aabo Akademi -Turku, Finland
Timothy Tsai, Hitachi Global Storage Technologies, USA
Sara Tucci-Piergiovanni, CEA List, France
Marco Vallini, Politecnico di Torino, Italy
Ángel Jesús Varela Vaca, University of Sevilla, Spain
Bruno Vavala, Carnegie Mellon University, USA | University of Lisbon, Portugal
Phan Cong Vinh, Nguyen Tat Thanh University, Vietnam
Hironori Washizaki, Waseda University, Japan
Byron J. Williams, Mississippi State University, USA
Victor Winter, University of Nebraska at Omaha, USA
Dong Xiang, Tsinghua University, China
Chun Jason Xue, City University of Hong Kong, Hong Kong
Hiroshi Yamada, Keio University, Japan
Liu Yang, Nanyang Technological University, Singapore
Piyi Yang, University of Shanghai for Science and Technology, China
Il Yen, University of Texas at Dallas, U.S.A
Hee Yong Youn, Sungkyunkwan University, Korea

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

A Fault-Injection Prototype for Safety Assessment of V2X Communication 1
Daniel Skarin, Benjamin Vedder, Rolf Johansson, and Henrik Eriksson

Fault-Detection Sensitivity Based Assessment of Test Sets for Safety-Relevant Software 5
Susanne Kandl and Jean-Marc Forey

Network-Security-Policy Analysis 10
Christian Pitscheider

Notification Support Infrastructure for Self-Adapting Composite Services 17
Erlend Andreas Gjaere, Per Hakon Meland, and Thomas Vilarinho

A Policy-based Middleware for Self-Adaptive Distributed Systems 25
Sun Jingtao and Satoh Ichiro

SLA Object and SLA Process Modeling using WSLA and BPM Notations,Towards defining a Generic SLA Orchestrator Framework 32
Bukhary Ikhwan Ismail, Nurliyana Muty, Mohammad Fairus Khalid, and Hong Hoe Ong

HiPAS: High Performance Adaptive Schema Migration - Evaluation of a Self-Optimizing Database Migration 41
Hendrik Muller, Andreas Prusch, and Steffan Agel

Evaluation of Software-Based Fault-Tolerant Techniques on Embedded OS's Components 51
Hosein Mohammadi Makrani, Amir Mahdi Hosseini Monazzah, Hamed Farbeh, and Seyed Ghassem Miremadi

A Fault-Injection Prototype for Safety Assessment of V2X Communication

Daniel Skarin, Benjamin Vedder, Rolf Johansson, and Henrik Eriksson
Department of Electronics
SP Technical Research Institute of Sweden
Borås, Sweden
e-mail: {daniel.skarin, benjamin.vedder, rolf.johansson, henrik.eriksson}@sp.se

***Abstract*— This paper describes an approach for injecting faults in ad hoc vehicle networks. A prototype fault injector, which makes it possible to investigate how a cooperative vehicle system behaves in the presence of communication errors, has been developed. The prototype shows a feasible way to use fault injection as technique to produce evidence for a safety case belonging to a cooperative automotive system.**

Keywords-fault injection, safety assessment, IEEE 802.15.4, V2X communication.

I. INTRODUCTION

In the past years, there has been a strong focus on functional safety in the automotive domain. In 2011, the standard ISO 26262 [1] was released, and currently the industry is adopting the development procedure to the standard. At the same time, automotive functions are getting more and more complex; autonomous and cooperative vehicles will soon move from prototypes to products. Safety assessment of cooperative systems will put requirements on evidence which show that communication failures are handled in a safe way. This paper shows a way to inject communication faults in cooperative systems as a technique to produce evidence for a safety case.

Cooperative vehicle systems cover a wide range of interdependence. Willke *et al.* [2] have suggested a taxonomy defining four type levels. On type levels 1 and 2, vehicles and infrastructure are exchanging information without being dependent on it to achieve a safe behavior. On type level 3, the functions rely on communicated information from other vehicles about motion and actuator states to ensure safe and/or efficient operation. On type level 4, applications use inter-vehicle communication to reach a common goal, e.g. driving in a road train (platooning). At least on the type levels 3 and 4, safety requirements will be allocated on the communication between the vehicles (V2V) and between the cars and the infrastructure (V2I).

According to the ISO 26262 standard, safety requirements shall be refined from top-level safety goals to the system components of the physical architecture. For safety-related cooperative functions, this implies that some safety requirements will be put on the V2V and V2I communication, respectively. Furthermore, the standard states what is needed to argue in order to fulfil verification of the safety requirements. For the higher integrity levels (ASIL C and D), it is required to use fault-injection techniques to show that safety mechanisms can handle all safety-relevant faults.

Fault injection in wireless communication used for transfer of safety-critical information in ad hoc vehicle networks needs further research. For computer systems (hardware and software) communicating via wires, there is a fairly long tradition of using fault-injection techniques and tools [3]. Alena *et al.* [4] have investigated how the fault tolerance of wireless sensor networks using IEEE 802.15.4 is affected by interference from other networks and multi-paths. Boano *et al.* [5] present a solution which produce repeatable and precise patterns of interference in wireless sensor networks. Malicious faults (attacks) and some natural faults in ad hoc networks can be assessed using the fault-injection platform developed by de Andrés *et al*. [6].

In this paper, a fault-injection prototype is described. The prototype is based on IEEE 802.15.4 since this standard is used for communication in the automotive and aerospace demonstrators of the KARYON project [7]. However, it is straightforward to adapt the concept to other techniques to be used in the automotive domain (IEEE 802.11p).

Section II introduces relevant fault models originating from functional safety standards. The section also explains how different failure modes can be emulated. Section III describes the fault injection prototype, and Section IV presents initial conclusions and future work.

II. FAULT INJECTION IN COMMUNICATION

A. Fault models

Standards for functional safety, such as ISO 26262 for road vehicles and the generic IEC 61508, list failure modes which are applicable for communication. Part 5 of ISO 26262 [1] lists failure modes for on-chip communication and data transmission. The failure modes for data transmission are applicable for wireless communication. IEC 61508-2 [8] lists identical failure modes for communication. Other important failure modes for communication are blocking access to communication channel [9] and asymmetric information [10]. Table 1 summarizes failure modes applicable for wireless communication.

Based on the diagnostic coverage that is claimed for a safety mechanism, ISO 26262-5 Table D.1 [1] lists failure modes that need to be analyzed. Failure modes for on-chip communication are described next.

Stuck-at failures are described as a continuous low or high signal at the pins of an element. They are applicable for elements which have a pin-level interface for data, control, address, and arbitration signals.

 ISBN: 978-1-61208-378-0

TABLE I. FAILURE MODES FOR COMMUNICATION

Failure mode	Interpretation
Message Corruption	The received data of a message is incorrect.
Message delay	A message is received later than expected by all, or some, receivers.
Message loss	A message is lost by all, or some, receivers.
Unintended message repetition	Receivers obtain two or more messages with the same information instead of one message.
Resequencing	Messages are received with incorrect sequence numbering.
Insertion of message	Receivers obtain a message that they did not expect
Masquerading (or incorrect addressing)	A sender transmit messages using an id of a different sender
Asymmetric information	Information from a single sender is received differently by receivers. It can also be that information from a sender is only received by a subset of the receivers
Blocking access to a communication channel	Prevents nodes from accessing the communication channel, similar to a babbling idiot.

The *direct current* fault model extends stuck-at failures with stuck-open, open, or high impedance outputs, and short circuits between signal lines. The analysis of the fault model is applicable for data, control, address and arbitration signals, but is mainly intended for main signals or on highly coupled interconnections.

When several devices are connected to a bus, arbitration is used to determine which device that controls the bus. *No arbitration* and *continuous arbitration* are mentioned as failure modes for on-chip communication in ISO 26262-5 [1]. *Time out* is mentioned in both IEC 61508 and ISO 26262, but neither standard describes the failure mode in more detail.

Soft errors are caused by ionizing particles, supply voltage noise, or cross-coupling between signal lines. The consequence is one or several bit-flips in memories or bus signals.

B. Emulating the Effects of Faults

The failure modes for wireless data communication can be emulated using a combination of jamming, packet injection, and packet sniffing. Jamming [5][11] is used to prevent one or several nodes from receiving or sending packets. Packet injection is used to insert additional, duplicated or corrupted messages in the wireless network. Packet sniffing allows the fault injection module to eavesdrop the wireless traffic in a non-intrusive manner. This is useful for logging and for triggering the injection of different failure modes.

Table 2 shows how different failure modes can be implemented by combining jamming and packet injection. For example, the effects of a message delay can be emulated by jamming to prevent nodes from receiving the original message, and then resending the original message with a delay. This assumes that we have a priori knowledge of the content of the message. Message losses are emulated by activating jamming when specific messages are being transmitted by a node.

Figure 1 and Figure 2 illustrate how failure modes for on-chip communication are emulated. The signal between two elements passes through a fault injection module which has the capability to modify the transmitted signal value. For most failure modes, such as soft errors, a faulty signal only relies on the value of the non-faulty signal as shown in Figure 1. For short-circuits between signals, however, the values of two or more signals are needed, as shown in Figure 2.

TABLE II. EMULATING FAILURE MODES USING JAMMING AND PACKET INJECTION

Failure mode	Jamming	Packet Injection
Message Corruption	x	x
Message delay	x	x
Message loss	x	
Unintended message repetition		x
Resequencing		x
Insertion of message		x
Masquerading (or incorrect addressing)		x
Asymmetric information	x	x
Blocking access to a communication channel	x	

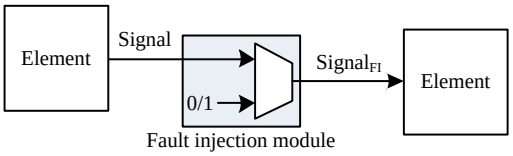

Figure 1. Injection of stuck-at faults in a signal.

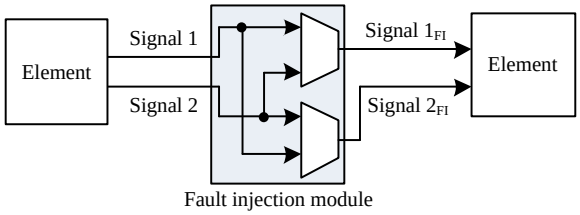

Figure 2. Injection of short-circuit failures between two signals.

C. Controlling When to Inject Faults

Figure 1 shows a state machine for controlling the fault injection. The idle state has an internal counter to keep track of the currently evaluated trigger. When all triggers have been evaluated to true in the correct order, fault injection is activated in the state "Start FI". Following that, the "FI" state is immediately entered.

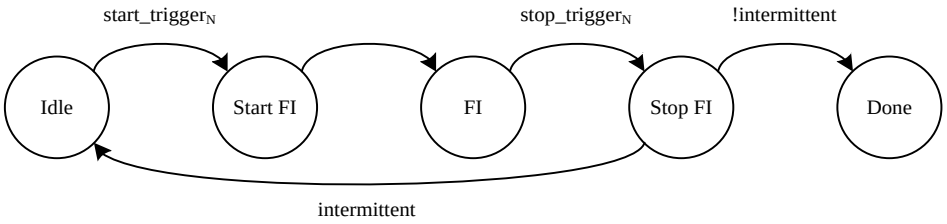

Figure 3. State machine to control the fault injection.

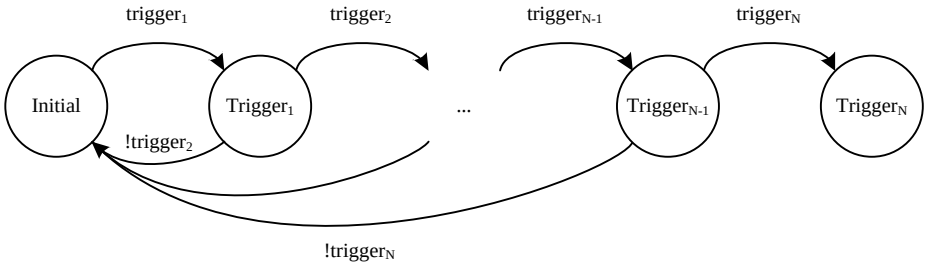

Figure 4. State machine to handle start and stop triggers for fault injection.

The "FI" state is exited when all stop triggers have been fulfilled. Unless an intermittent fault is emulated, the fault injection is stopped. For intermittent faults, there is a return to the idle state and another wait for start trigger fulfillment. Fault injection is activated using triggers which can be based on: elapsed time, probability per received packet, sender or receiver address of a packet, or data in the payload of a packet. Several triggers can be combined so that fault injection is started or stopped by a chain of events, as shown in Figure 2. Using this approach, well-known packet loss models such as Bernoulli and Gilbert-Elliot [12] can be supported, as well as simple triggers based on, e.g., elapsed time.

III. FAULT INJECTION PROTOTYPE

The fault injection concept described in the previous section has been implemented for vehicle demonstrators in the KARYON project [7]. The fault injection prototype can be used for injecting failures in IEEE 802.15.4 data communication, and in the on-chip communication. Figure 5 shows a picture of the fault injection node, which uses the STM32F4 microcontroller from ST and the CC2520 communication chip from Texas Instruments. The node is based on layout and hardware schematics which are freely available from [13].

The fault injector uses ChibiOS/RT [14] as its operating system, and implements the state machine described in Section II.C. The following fault injection triggers are supported:

- Time – Enabled after a specified time has elapsed.
- Packet probability – Enabled with a specified probability for each received packet.
- Packet destination address – Enabled when a packet with a matching source address is received.
- Packet source address – Enabled when a packet with a matching destination address is received
- Packet data – Enabled when the specified data matches the received data
- The fault injector is configured using USB commands, or by sending configuration packets via IEEE 802.15.4.

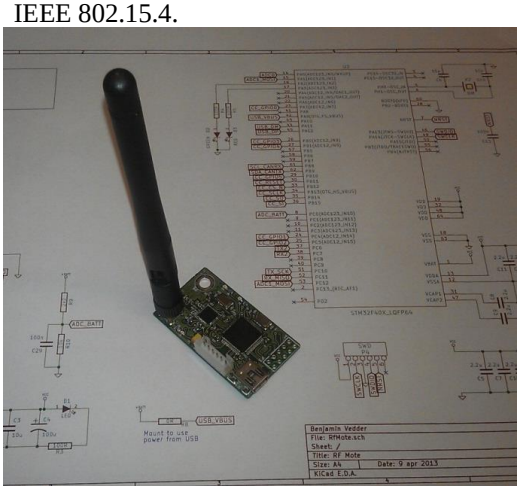

Figure 5. RF board with STM32F4 and CC2520 based on [Vedder].

The prototype fault injector also provides packet logging capabilities, which are useful for debugging purposes. The CC2520 communication chip provides hardware support for packet sniffing, which can be used as a non-intrusive method of observing wireless traffic. The fault injector can output captured packets in the packet capture (pcap) format using a named pipe. The logged traffic can then be analyzed in real-time using tools such as Wireshark which is an open source network protocol analyzer. Figure 6 shows an example of logged traffic in Wireshark.

The following failure modes are currently supported by the fault injection prototype: message corruption, delay, loss, insertion, unintended message repetition, masquerading, and blocking access.

 ISBN: 978-1-61208-378-0

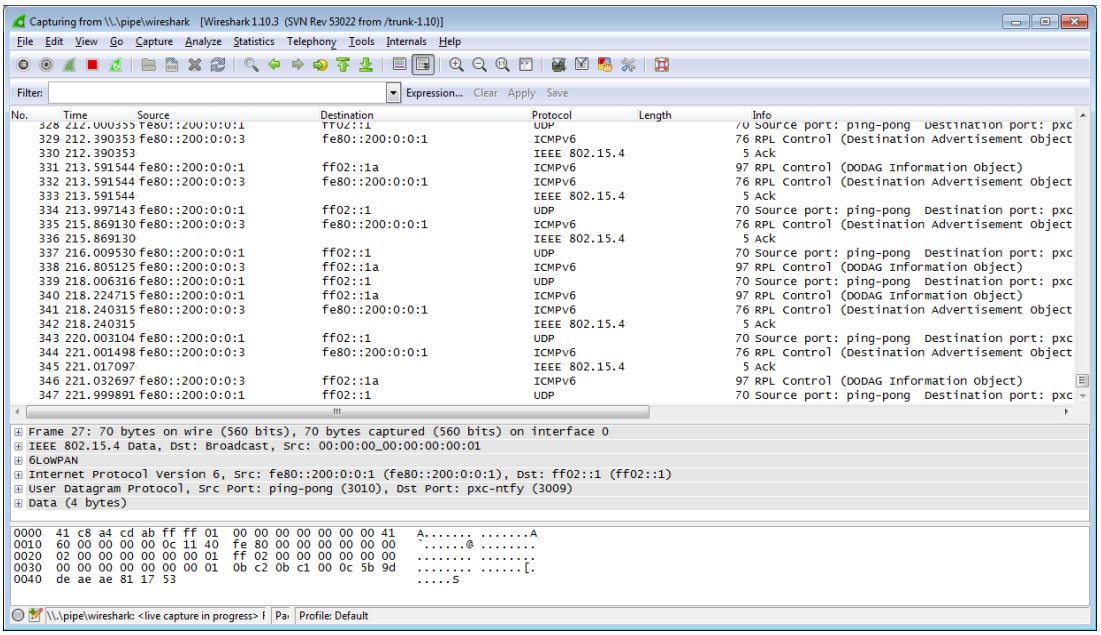

Figure 6. Packet sniffing using the fault injection node and Wireshark.

For some of the failure modes, e.g. message delay, message payload need to be known a priori. Proof-of-concept fault injections have been successfully performed, but no complete fault-injection campaigns have been run yet.

IV. CONCLUSIONS AND FUTURE WORK

A prototype fault injector for digital communication, in particular wireless communication, has been described. One limitation with the approach is that communication chips require some time to switch between receiving and sending. For the CC2520 chip, the RX/TX turnaround time is 192µs. For packets with a small payload, it might therefore not be possible to trigger the fault injection and jam the packet currently being received. This is something which will be investigated in the near future.

The prototype has been tested on IEEE 802.15.4 communication, but the concept is straightforward to adapt to other communication techniques, such as IEEE 802.11p.

The prototype shows that it is feasible to inject most faults needed in a safety assessment according to the requirements in functional safety standards.

ACKNOWLEDGMENT

This work has been supported by the EU under the FP7-ICT program, through project 288195 Kernel-based ARchitecture for safetY-critical cONtrol (KARYON).

REFERENCES

[1] ISO26262-5, “Road vehicles – Functional safety – Part 5: Product development at the hardware level”, 2011.

[2] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk, “A survey of inter-vehicle communication protocols and their applications”, IEEE Communications Surveys & Tutorials, vol. 11(2) , pp. 3-20, 2009.

[3] H. Mei-Chen, T. K.Tsai, and R. K. Iyer, “Fault injection techniques and tools”, IEEE Computer, vol. 30(4), pp. 75-82, 1997.

[4] R. Alena, R. Gilstrap, J. Baldwin, T. Stone, and P. Wilson, “Fault tolerance in ZigBee wireless sensor networks”, Proc. 2011 IEEE Aerospace Conference, March 2011, pp. 1-15.

[5] C. A. Boano et al., “Controllable radio interference for experimental and testing purposes in wireless sensor networks,” Proc. IEEE 34th Conference on Local Computer Networks, Oct. 2009, pp. 865-872.

[6] D. de Andrés, J. Friginal, J.-C. Ruiz, and P. Gil, ”An attack injection approach to evaluate the robustness of ad hoc networks”, Proc. 15th IEEE Pacific Rim International Symposium on Dependable Computing, Nov. 2009, pp. 228-233.

[7] Homepage of Kernel-Based ARchitecture for safetY-critical cONtrol, http://www.karyon-project.eu/, accessed on 25th of June 2014.

[8] IEC 61508-2, “Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems”, 2010.

[9] H. Kopetz, “Real-time systems”, Kluwer, 1997.

[10] F. Cristian, “Understanding fault-tolerant distributed systems”, Communications of the ACM, vol. 34, pp. 56-78, 1991.

[11] A. D. Wood, J. A. Stankovic, and G. Zhou, "DEEJAM: Defeating energy-efficient jamming in IEEE 802.15. 4-based wireless networks," Proc. 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'07), June 2007, pp. 60-69.

[12] J.-P. Ebert and A. Willig, “A Gilbert-Elliot bit error model and the efficient use in packet level simulation”, Technical Report, TKN-99-002, Technical University of Berlin, 1999.

[13] Homepage of B. Vedder “CC2520 and STM32 RF boards”, http://vedder.se/2013/04/cc2520-and-stm32-rf-boards/, accessed on 25th of June 2014.

[14] Homepage of ChibiOS/RT, http://www.chibios.org, accessed on 25th of June 2014.

 ISBN: 978-1-61208-378-0

Fault-Detection Sensitivity Based Assessment of Test Sets for Safety-Relevant Software

Susanne Kandl
Institute of Computer Engineering
Vienna University of Technology
Austria
Email: susanne@vmars.tuwien.ac.at

Jean-Marc Forey
Synopsys, Inc.
Grenoble
France
Email: Jean-Marc.Forey@synopsys.com

***Abstract*—In testing it is, in general, not possible (or at least extremely time-consuming) to cover the complete input data space, therefore usually a test set is selected for a given coverage criterion (like decision coverage or similar). This restricted test set covers only a part of the complete input data space with a degraded fault-detection capability. The fault-detection capability of a test set is given by the number of *detected* faults in relation to the number of *actual* faults. In this work, we present a novel approach for the assessment of test sets based on their fault-detection sensitivity. The main goal of an efficient testing process is to reduce the test effort while targeting a maximum number of detected faults, i.e., an ideal test set requires a minimal execution time for the test run (defined by the number of the test cases and their individual run-times) with a maximum fault-detection sensitivity. Therefore, our proposed test-set selection process is not guided by a coverage criterion, but by the fault-detection capability of the different test cases using the output of the tool *Certitude Functional Qualification System*. We will apply our strategy on a safety-relevant (regarding ISO 26262) case study from the automotive domain focusing on the scalability of the test-set selection strategy. Our work presents a novel method to optimize the testing effort for software used in dependable systems with respect to a decreased testing effort while sustaining a high fault-detection capability.**

Keywords–Dependable Systems, Testing, Fault-Detection Sensitivity, Safety-Relevant Software.

I. INTRODUCTION

The Verification and Validation (V&V) of safety-relevant systems (a class of dependable systems) requires a tremendously high amount of effort [1]. Besides techniques like analysis or review, testing is one of the main methods to prove the correctness of the system. The project *Verification and Testing to Support Functional Safety Standards* (VeTeSS) [2] deals with the development of standardized tools and methods for the verification of safety-relevant systems in the context of ISO 26262 (Functional Safety for Road Vehicles) [3]. One main aspect is to tackle the contradiction between *reducing the V&V-effort* while *enhancing the dependability* of the safety-relevant components. Usually, there is a correlation between the testing effort and the confidence in the test process (the test result, respectively) illustrated in the exemplary Figure 1. An increasing testing effort results in a higher confidence in the test process. After passing a critical mark in the testing effort (indicated by the dashed line), the confidence in the testing result increases only marginally approaching 100% confidence in an asymptotic way for additional testing effort.

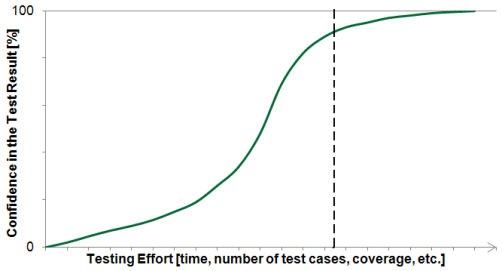

Figure 1. Correlation Between the Testing Effort and the Confidence in the Test Process

The selection of the test set (the set of test cases) has a significant impact on the testing effort and the resulting test result. The example given in Figure 2 demonstrates this fact (TD means Test Data: some input data to execute this branch; TC stands for Test Case: an execution trace in the program). The control flow of this example consists of 4 if-decisions. Each decision has 2 branches, i.e., in the overall we have to cover 8 branches for 100% Decision Coverage (DC). For that we need test cases to cover all the branches (b_1 + b_2, b_3 + b_4, b_5 + b_6, b_7 + b_8). Due to the redundancies in the traces this results in 5 test cases (TC_1, TC_2, TC_3, TC_4, TC_5). Starting with Test Set TS_{INI} consisting of TC_1 and TC_5 shows that 4 out of 8 branches are covered, that means we achieve 50% DC. Adding the test case TC_2 to TS_{INI} results in covering 5 of the 8 branches, this equals 62,5% DC (i.e., an increase of 12,5%). Adding the test case TC_3 to TS_{INI} results in covering 6 of the 8 branches, this equals 75,0% DC (i.e., an overall increase of 25,0%). That means that both extended test sets contain 3 test cases, but the latter one achieves better results regarding the coverage criterion DC. Although TC_3 is longer than TC_2 (incorporating more test steps), usually the length of the test cases has a negligible effect on the testing effort, as the initialization of the test cases is the most time-consuming part in the test-case execution.

The assessment of a test set based on a given coverage criterion is a very common technique but studies (like [4] [5] [6]) indicate that structural coverage is a rather insufficient means to determine the quality of the test set. Even sophisticated coverage criteria like MC/DC (Modified Condition/Decision Coverage) do not guarantee a high error-detection sensitivity (see [7] [8]). What we are *really* interested in is the *fault-detection sensitivity* of a test set, that means the number of *detected* faults in relation to the number of *actual* faults for

 ISBN: 978-1-61208-378-0

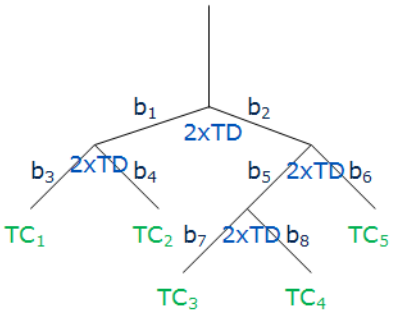

Figure 2. Example for Test-Set Selection for Decision Coverage

a given test set. Faults are erroneous parts in the program (colloquially aka: bugs), for instance, a mutated name of a variable (`variable_1` instead of `variable_2`). This motivated the idea of a fault-detection sensitivity based assessment of test sets. The main principle is to select the test set on the basis of the powerfulness of test cases to detect faults. For this we are using the analysis of the Certitude Functional Qualification System [9]. This tool introduces artificial faults into a program and determines which faults are covered by specific test cases. Some of the test cases are capable to reveal a high number of faults, whereas other test cases detect only a few faults. The target is to assess a test set regarding the fault-detection sensitivity (given by the capability of the test cases to detect faults). To evaluate the feasibility of this idea for a real industrial case study we apply our strategy to a case study from the automotive domain (software for the selection of the driving mode for an electric car).

In the remaining part of the paper, we give some basic definitions (Section II) and describe the functionality of the tool Certitude to explain how we are using this tool for our approach (Section III). In the following, we elaborate our strategy for fault-detection sensitivity based assessment of test sets and present the questions addressed by our study (Section IV). Finally, we give a short overview on our work in progress to evaluate our approach on a case study (Section V) and conclude with a summary of the paper and planned future work (Section VI).

II. Basic Definitions

A **Test Set** TS consists of *test cases*. A **Test Case** TC is a *trace* in the execution of the program defined by the variable values *including* the correct output for given input data. Example: A test case for a function `sum(a, b)` may be $(2; 3; 5)$. The **size** of a test set $S(TS)$ is defined by the number of test cases.

A **fault** is a mutation of the program (i.e., a deviation of the correct implementation). We will consider faults for operators, variables, and values.

A **strong** test case is a test case that is capable to detect many faults. A **weak** test case is a test case that is capable to detect only a few faults. The fault-detection sensitivity of a test case $F(TC_x)$ is defined by the number of faults that are detected by this test case. The fault-detection sensitivity of a test set $F(TS)$ is defined by the sum of the detected faults of the test cases in TS, i.e., $F(TS) = \sum_{x=1}^{S(TS)} F(TC_x)$.

The classification of test cases into *strong* and *weak* test cases depends on the achieved values for $F(TC_x)$ for the test cases in TS: A test case with $F(TC_x) = 10$ may be a *strong* test case in a test set with test cases with a maximum fault-detection sensitivity of 12, but it is rather a *weak* test case when other test cases in this test set are able to detect, e.g., 100 faults.

In the following, we will also distinguish between an **Easy-To-Detect (ETD) fault** and a **Not-Easy-To-Detect (NETD) fault**. Whereas the first one is detected by many test cases (for instance, 27 out of 100 test cases), the second one is only detected by a few test cases (for instance, 2 out of 100 test cases). We define the number of test cases of a test set that are capable to detect a specific fault (Ftl_y) $D(Ftl_y) = \#TC_x$ (for $x = 1...S(TS)$) with TC_x detects the fault Ftl_y, as the metric to guide the classification of faults. The precise distinction between *EDT* and *NEDT* faults depends on the concrete values for a case study (the program under test and the applied test set).

III. Certitude Functional Qualification System

Basically, the tool Certitude Functional Qualification System by Synopsys Inc. [9] intends to measure the effectiveness of a verification environment. It is able to identify verification weaknesses that allow bugs to stay undetected in the testing process and may lead to functional problems. The basic principle of Certitude is to introduce mutations (i.e., artificial software faults) into the design and prove whether the verification environment (the test set) is capable to detect these faults or not, see Figure 3 (RTL refers to Register-Transfer Level). Please, consider that this technique is different to typical software fault injections with the intention to validate fault-handling mechanism at runtime and to evaluate the way a system behaves in the presence of faults (like [10]). The aim of Certitude is to determine the capability of a verification environment to detect design mutations (similar to [11] and [12]).

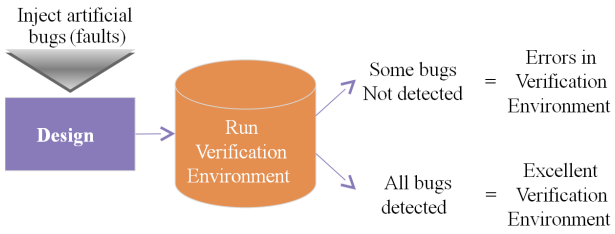

Figure 3. Certitude Functional Qualification Process

If all (or at least most) of the artificially introduced bugs are identified by the verification environment, this V&V-environment is proven to be *good*. If many of the artificially introduced bugs are *not* identified by the verification environment, this indicates that the V&V-environment is insufficient. Either some test cases (or specific test scenarios) are missing or wrongly implemented, or the checks are not robust enough (missing or broken). Certitude provides two different modes: 1. The *verification improvement mode* analyzes the verification of the design and identifies specific holes and weaknesses. 2. The *metric mode* allows to measure the overall quality of the verification environment in a quantitative way using a statistical approach.

 ISBN: 978-1-61208-378-0

Certitude combines static analysis with mutation-based techniques for introducing mutations (i.e., artificial bugs) into the system under test, see Listing 1 for an example. In line 1, the original version is given. Then the Boolean operator *OR* (|) is mutated to the Boolean operator *AND* (&) resulting in the faulty version in line 2.

```
1    a = b | c;
2    a = b & c;
```

Listing 1. Original Code and Faulty Program Code

After introducing the mutations, Certitude determines whether the V&V-environment can activate (i.e., exercise) the faulty code, propagate the effects to an observable point, and detect the presence of the fault. This is done in three phases (see Figure 4): a) In the fault model analysis phase, the design is analyzed and appropriate faults are selected that will be injected into the system. b) In the fault activation phase, the specified tests (selected from the regression) are run once and the behavior of the V&V-environment with respect to the faults is analyzed. c) In the fault detection phase, selected test cases from the V&V-environment are executed to measure the ability of the V&V-environment to detect the faults.

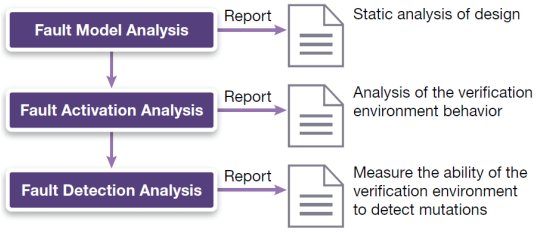

Figure 4. Phases of the Certitude Functional Qualification

Furthermore, Certitude uses a proprietary algorithm to automatically classify and prioritize the faults (the injected faults are qualified in a priority order). The subsequent qualifications contain the results from the previous test runs and focus on the remaining undetected faults. By this, it is possible to find and fix weaknesses in the V&V-environment in an early stage of the verification process, expand the set of qualified faults (as both, the V&V-environment and the design, evolve), achieve an incremental improvement over time, and minimize the effort for analysis and debug. *Latent* faults are mutations that have no impact on the program behavior. Some of them can be recognized by Certitude in the diagnosis of the results of the test run. At the moment, Certitude supports hardware description languages, like VHDL [13] and Verilog [14], and programming languages like C/C++.

Within the analyses of Certitude the tool provides a list that shows for each executed test case the number of *activated*, *propagated*, and *detected* faults. The code skeleton given in Listing 2 demonstrates the meaning of the three different terms: In line 2, the correct program version is given (with `i<11`) and in line 3, the mutated (faulty) program version is given (with `i<=11`). For the activation of this fault we need a test case with the variable `a` equals the value *TRUE*, otherwise the fault is not activated at all. The propagation of a fault to an observable failure is necessary to observe (and thus, detect) the actual fault. A fault may cause an error (an invalid state in the system behavior). An error may cause further errors (therefore an error may act as a fault), or it may propagate and then be observable. To propagate the injected fault, we need a test case with the value 5 for the variable `j`. Only then we are able to observe a deviation from the original program behavior caused by the introduced fault.

```
1    while (a == TRUE) {
2       for(i=1; i<11; i++) {    //correct version
3       for(i=1; i<=11; i++) {   //mutation
4          print (i);
5          if (j==5) {
6             product = i * j;
7             print (product);
8          }
9       }
10      if (product > 50)
11         print (error_message);
12      else
13         print (OK);
14   }
```

Listing 2. Original Code and Faulty Program Code

The main output of Certitude for the quality assessment of the V&V-environment (the underlying test set, respectively) is the number of detected and undetected faults. If the ratio of detected faults to the overall number of injected faults is high, this indicates a matured and reliable test set, otherwise the test set has to be improved. For further analysis, Certitude also provides a list showing the number of activated (*#ActF*), propagated (*#PropF*), and detected faults (*#DetF*) for each test case, see the exemplary Table I.

TABLE I
EXAMPLE OUTPUT OF CERTITUDE

Test Case	*#ActF*	*#PropF*	*#DetF*
TC_1	80	70	50
TC_2	60	50	20
TC_3	20	10	5
etc.	etc.	etc.	etc.

Assuming a total number of introduced faults of 100, TC_1 appears to be a *strong* test case, whereas TC_3 is a rather *weak* test case regarding the lower number of detected faults.

The main purpose of the Certitude Functional Qualification System is to provide the information that allows the test engineer to improve the overall V&V-environment (i.e., the test set) by identifying weaknesses of the V&V-environment (e.g., missing test cases) and improving the V&V-environment (by adding these missing test cases). In the following, we present an idea to use the tool Certitude for the selection of test sets that require less testing effort while sustaining a high fault-detection sensitivity.

IV. FAULT-DETECTION SENSITIVITY BASED ASSESSMENT OF TEST SETS

The main idea of our strategy for the selection of a test set is to include the test cases with a high fault-detection sensitivity $F(TC_x)$ and to dismiss the test cases with a low $F(TC_x)$. By this, we achieve a smaller test set (thus, reducing

 ISBN: 978-1-61208-378-0

the overall test effort). The fault-detection sensitivity based assessment of test sets is instrumented by a) a parameter for the number of detected faults by the definition of a threshold-value T_{DetF} for detected faults: if $F(TC_x)$ is greater than T_{DetF}, then TC_x is included in the final test set otherwise not. or b) a parameter for the targeted reduction of the test set by defining a concrete value for the reduction (e.g., 20% of the original test cases are omitted).

In this study, we are mainly interested in:

- What is the exact effect on the degradation of the fault-detection sensitivity of the new test set $F(TS)$ for a) and b)? It may happen that a test case with a low $F(TC_x)$ is discarded without any (negative) impact on $F(TS)$ because the related faults are anyway covered by other (remaining) strong test cases. On the other hand omitting a weak test case may have a significant impact on $F(TS)$ because this test case is required for the activation of a couple of faults (without activation of these faults they cannot be detected by the strong test cases).
- What is the relation between the achieved reduction regarding the test effort (correlating with the size of the test set) and the resulting degradation of the fault-detection sensitivity of the test set $F(TS)$? If we risk to fail to detect important faults just by omitting 5 out of 1000 test cases (i.e., the effort reduction is almost negligible), then the efficiency of the strategy is questionable. Preliminary empirical results indicate that omitting some of the weak test cases results in a significant reduction of the test effort while degrading the actual fault-detection sensitivity $F(TS)$ only marginally.
- Experiences so far suggest that for our analysis we have to differ between *Easy-To-Detect (ETD)* faults and a *Not-Easy-To-Detect (NETD)* faults. What does a low/high number of ETD-/NETD-faults means for our strategy? For a system with many ETD-faults, in general, the omission of a weak test case has only a tiny effect on the overall $F(TS)$. This is not valid for NETD-faults. Certitude also provides implicitly some information to distinguish between ETD- and NETD-faults. This information can support the evaluation of our strategy.
- Is our strategy (due to computation time) feasible for a real industrial case study? In general, determining a minimal test set is decidable but, the problem is NP-complete (that means the algorithm requires an exponentially high effort), thus usually heuristics are applied to find an optimal test set [15]. We try to avoid this problem by only a few test cycles to determine $F(TS)$ and the values for $F(TC_x)$ for the initial test set. Selecting the test cases for the improved test set is linear and then we are executing the test runs for the reduced test set.

V. Evaluation on an Industrial Use Case

In our empirical evaluation, we focus on automotive software written in the programming language C. One of our industry partners provides a use case for an automatic gear selection for an electric car. The Electric Vehicle Controller (EVC) manages the functions for the gear selection of the available states (for instance: Park; Reverse; Neutral; and Drive, the function similar to automated transmission). The main function of the use case allows the selection of the driving mode from the four different available states. The initial test set is generated automatically from a SysML-model of the use case. With the final results of the empirical evaluation of our strategy for fault-detection sensitivity based assessment of test sets, we will be able to give clear recommendations for the applicability, the benefits, and also the limitations of our novel idea.

VI. Conclusion and Future Work

Finding the optimal test set (minimal size and maximum fault-detection capability) is a permanent ongoing challenging task. Experienced test engineers may be able to write down a few good test cases (good in finding faults), some of them especially tuned to detect the faults/errors of a certain program developer or adapted to intricate (thus error-prone) parts of the system under development. Coverage metrics are often used to determine the maturity of the test set based on the assumption that an increase in the coverage induces a better fault-detection sensitivity. In this work we present a novel idea for the assessment of the quality of a test set by the main attribute we are interested in, namely the *fault-detection sensitivity*. For our strategy, we made use of the tool Certitude Functional Qualification System that provides us with the necessary information about the fault-detection capability of the different test cases of a test set. Static analysis combined with mutation-based fault injection yields in precise information about the powerfulness/weakness of a test case to detect faults. This information can be used for a qualified selection of test cases to reduce the size of the test set while preserving a desired test set quality (regarding the fault-detection sensitivity). Our idea intends also to motivate a test process stronger guided by fault injection instead of using fault injection only as a recommended method prescribed by standards for safety-relevant systems, like ISO 26262.

Besides finishing the analysis for the mentioned use case, we plan to extend the evaluation of our strategy to (at least) another case study. Not only the number, but also the type of faults and the program structure influence significantly the fault-detection sensitivity of a test set, so precise answers to the listed questions can only be given after results from different use cases. We also consider to address faults leading to a multiple point failure (individual faults may lead only in combination with another independent fault/with other independent faults to a failure, the so-called multiple point failure). This kind of faults are usually difficult to detect (as they are only observable in the presence of other faults).

Acknowledgments

This work has been partially funded by the ARTEMIS Joint Undertaking and the National Funding Agency of Austria for the project VeTeSS under the funding ID ARTEMIS-2011-1-295311.

 ISBN: 978-1-61208-378-0

REFERENCES

[1] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, 2nd ed., ser. Series: Real-Time Systems Series. Springer, 2011.

[2] "ARTEMIS VeTeSS: Verification and Testing to support functional Safety Standards," 2014, last visited: 09-22-2014. [Online]. Available: http://www.vetess.eu

[3] ISO: International Organization for Standardization, "ISO 26262: Functional safety – road vehicles," 2011.

[4] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering, ser. FASE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 409–424, last visited: 09-22-2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28872-2_28

[5] K. Kapoor and J. Bowen, "Experimental evaluation of the variation in effectiveness for DC, FPC and MC/DC test criteria," Proceedings International Symposium on Empirical Software Engineering, ISESE 2003, Sept.-1 Oct. 2003, pp. 185–194.

[6] J. Guan, J. Offutt, and P. Ammann, "An industrial case study of structural testing applied to safety-critical embedded software," in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ser. ISESE '06. New York, NY, USA: ACM, 2006, pp. 272–277, last visited: 09-22-2014. [Online]. Available: http://doi.acm.org/10.1145/1159733.1159774

[7] A. Dupuy and N. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software," in Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th, vol. 1, 2000, pp. 1B6/1–1B6/7 vol.1.

[8] S. Kandl and R. Kirner, "Error detection rate of MC/DC for a case study from the automotive domain," LNCS 6399: S.L.Min et al.(Eds.): Proceedings of the 8th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2010), Oct. 2010, pp. 131–142.

[9] Synopsys Inc., "Certitude - functional qualification system," 2014, last visited: 09-22-2014. [Online]. Available:https://www.synopsys.com/TOOLS/VERIFICATION/FUNCTIONALVERIFICATION/Pages/certitude-ds.aspx

[10] D. Alexandrescu, L. Sterpone, and C. Lopez-Ongil, "Fault injection and fault tolerance methodologies for assessing device robustness and mitigating against ionizing radiation," in Test Symposium (ETS), 2014 19th IEEE European, May 2014, pp. 1–6.

[11] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," IEEE Trans. Softw. Eng., vol. 32, no. 9, Sep. 2006, pp. 733–752, last visited: 09-22-2014. [Online]. Available: http://dx.doi.org/10.1109/TSE.2006.92

[12] J. Duraes and H. Madeira, "Emulation of software faults: A field data study and a practical approach," Software Engineering, IEEE Transactions on, vol. 32, no. 11, Nov 2006, pp. 849–867.

[13] P. J. Ashenden, The Designer's Guide to VHDL, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[14] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2003.

[15] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," ACM Trans. Softw. Eng. Methodol., vol. 2, no. 3, Jul. 1993, pp. 270–285, last visited: 09-22-2014. [Online]. Available: http://doi.acm.org/10.1145/152388.152391

 ISBN: 978-1-61208-378-0

Network-Security-Policy Analysis

Christian Pitscheider

Dip. di Automatica e Informatica
Politecnico di Torino
Torino, Italy
christian.pitscheider@polito.it

***Abstract*—Computer network security is the first line of defence to accomplish information assurance. The computer network is at risk without a well-designed and flawless implemented network security policy. The main problem is that network administrators are not able to verify the network security policy. Although further research has been carried out, it mainly concerns small specific parts of the overall problem. This paper presents different approaches from literature and highlights how they are correlated and can operate together. This work summarizes the solutions proposed in literature, points out their advantages, disadvantages and limitations. To conclude, it proposes solutions for future research in this area.**

Keywords–Security Policy; Analysis; Reachability; Policy comparison.

I. INTRODUCTION

More and more computer networks are connected to the Internet and remote sites are becoming more frequent. As a result, computer networks have become a very complex structure that is hard to manage. As some studies have shown, firewall configuration errors are quite frequent [1][2]. The studies point out that network administrators do not have a good insight of the network and its configurations. Dedicated tools and procedures are needed to support the daily work of network administrators.

In literature, different approaches exist to help network administrators in their daily workflows. In general, the approaches can be divided into two distinct categories: policy analysis and policy generation. Policy analysis focuses on existing and deployed configurations, while policy generation focuses on automatically generating new configurations.

This survey gives a brief overview of policy generation but its main focus is on policy analysis, and in particular on three distinct policy analysis categories, namely, Conflict analysis, Reachability analysis, and Policy comparison.

The main limitation of the papers concerning policy analysis is that they focus on one single type of security control and cannot be applied to a complex computer network. For example, Al-Shear proposes a solution to perform conflict analysis of firewall policies [3]; however, this solution is not able to model Network Address Translation (NAT) / Network Address and Port Translation (NAPT) devices; therefore, computer networks that include NAT/NAPT devices cannot be analysed with this solution.

Another limitation is that the solutions are not compatible with each other. A model of a security control used in one approach cannot be reused in another. This means that a lot of research time is lost on modelling various security controls for each approach. For example, the reachability analysis model in [4] can also handle NAT devices; but, since the analysis model of this solution is not compatible with the one in [3], the model of NAT/NAPT devices cannot be reused and a new model must be defined to support this type of security control.

This paper first gives an extended overview of research carried out in this field and highlights the advantages, disadvantages, and limitations. Based on this analysis, we show that future research in this area should be concentrated on a unified analysis model. We also discuss what features this model should include and why such a model is desirable.

The rest of the paper is organized as follows. Section II presents the theoretical background. Section III presents a typical workflow that the network administrators may use to configure firewalls. Section IV presents the research carried out on different types of policy analysis techniques. Section V presents the summary of what is missing in the literature and how to fill the gap. Section VI concludes and summarizes the paper.

II. TECHNICAL BACKGROUND

A. Network Security Policy

A *network security policy* is a special kind of policy that focuses on security aspects of a computer network. Network security policies can be written in different formats and at different levels of abstraction. On the one hand, very abstract high-level policies exist which are written in natural language, that express network-wide security goals. On the other hand, concrete configuration of single security controls are written in a device-specific configuration language. High-level policies are easy to write and understand by humans but difficult to elaborate on machines; concrete configurations which are difficult to read and write for humans are easily interpreted by machines.

B. Security controls

Security controls are appliances or software modules of appliances within a computer network. They implement the functionalities needed to enforce a network security policy. Security controls can control the network traffic by blocking certain packets or modifying it by changing header information of certain packets. As an example, packet filters, stateful firewalls, and application-level firewalls are used to control

 ISBN: 978-1-61208-378-0

the traffic, whereas IPsec gateways, Virtual Private Network (VPN) terminators, and NAT/NAPT devices are able to modify the traffic.

C. Policy Analysis

Each of the three main policy analysis types focuses on a part of the analysis process, but they have overlapping functions and common steps to reach their goal.

Conflict analysis searches for possible errors within a single or a set of security policies. It searches for potential semantic errors within correlated policy rules. Conflict analysis can also be used to identify possible policy optimizations. Conflict analysis can be applied to a single policy (Intra-Policy analysis) or to set of policies of interconnected security controls (Inter-Policy analysis).

Reachability analysis evaluates allowed communications within a computer network. Furthermore, it can determine if a certain host can reach a service or a set of services. In general, reachability analysis is performed online by using tools such as "ping" or "traceroute". By using an accurate representation of the network and its security policies, reachability analysis can also be performed offline, during the design phase.

Policy comparison compares two or more network security policies and represents the differences between them in an intuitive way. Network security policies involved may include single concrete security control configurations, sets of configurations, and high-level policies of an entire network. One of the best use-cases of policy comparison is to verify that a desired network security policy is implemented correctly by comparing the designed high-level policy with the concrete network configuration.

III. Network Administrator work-flows

Research efforts in this field can be divided into two main groups: policy generation that proposes a complete new approach to policy definition, and policy analysis that tries to give additional support to already deployed systems.

The policy generation approach forces network administrators to completely redefine their workflow and to use less expressive configuration interfaces. Policy generation has the disadvantage that administrators cannot rely anymore on their previous work experience and have to trust a black box policy generation tool. Probably the bigger disadvantage is that already deployed systems cannot be integrated seamlessly, instead they have to be reconfigured from scratch by mean of policy generation tools.

The policy analysis approaches, works on already deployed devices, thus it has the advantage that administrators can continue their usual work and use policy analysis support only during complex tasks. Deployed systems remain unchanged and under the complete control of network administrators.

A. Policy generation workflows

The policy generation approach consists of three main parts: a high-level security policy, a model of the network topology and a policy refinement tool. The network administrator specifies the desired network security policy using a high-level language and abstractly represents the target network topology. The high-level security policy and the network representation are the input for the policy refinement tool. The transformation process implemented by the tool produces the device-specific configuration files.

The advantages of such approaches are limited by the expressiveness of the high-level policy, the number of supported device types and device manufactures, and the optimization that the transformation process introduces to the final configuration.

B. Policy analysis workflows

The application of policy analysis solutions proposed in literature follow specific workflows. Conflict analysis searches for potential errors within a configuration. Reachability analysis allows the administrators to query if specific properties of the configuration are true. Last but not least, policy comparison helps network administrators to identify differences between policies.

For a complete workflow from the design phase to implementation, testing and maintaining a network policy, all three analysis approaches must be applied. First, during the design phase of a policy, network administrators express the desired network security policy in a high-level language. Since at the moment there are no enterprise grade transformation tools to transform high-level policies into device specific configurations, administrators have to create the configurations by hand. The next step is to use a conflict analysis tool to identify potential errors, performance issues and rules which are never applied. After having reduced potential errors and performance issues, administrators may use a reachability analysis tool to verify that the key aspects of the desired policy are applied correctly. The last step is to use a policy comparison tool to compare the desired network security policy with the newly created one.

Other activities can be performed after the configurations have been deployed. Administrators may want to troubleshoot a connection problem using the reachability analysis tool. Having pinned down the connection problem to a missing firewall rule, the administrator wants to verify that a modification of this firewall configuration does not introduce conflicts and that only the desired change is applied. First, he uses the conflict analysis tool and afterwards he uses the policy comparison tool to compare the original with the modified configuration.

IV. State of the art

A. Policy refinement

In literature, different approaches exist towards automatic policy generation. Even though they show a great potential, the research is still ongoing and has not been adopted widely. Only a few enterprise grade products exist which have implemented such features and the adoption rate is fairly low. Algosec, the leader in network security management, has only a few thousand costumers. According to one of their surveys [5], only 13.4% of network operators use a centralized policy management whereas 74.8% do not use any type of automated tools. The survey considers as centralized policy management any type of policy analysis and policy refinement.

 ISBN: 978-1-61208-378-0

Bartal et al. propose a solution named Firmato [6]. It was one of the first solution proposals in this area and supports only packet filter firewalls. It is based on an entity-relationship model of the security policy and of the network topology. The entity-relationship model is compiled and translated into firewall specific configuration files. The prototype was used to manage a real network containing a single firewall with 50 rules.

Verma et al. [7] used a similar approach; the authors present a firewall analysis and configuration engine named FACE. It takes as input the network topology and a global security policy written in a high level language. FACE has two advantages over Firmato: firstly it can also analyse the firewall configurations created and secondly it configures only one secure path between source and destination instead of inserting ACCEPT rules on every possible path.

Garcia-Alfaro et al. [8] proposed MIRAGE, a management tool for the analysis and deployment of configuration policies. It is based on the same principles as Firmato [6] and FACE [7], but it is also capable of configuring intrusion detection systems IDS and VPN routers. MIRAGE can also perform policy analysis on already deployed configurations.

Casado et al. [9] take a different approach; they proposed a solution named SANE. Instead of generating concrete configurations for already deployed firewalls, it proposed a new architecture where the network contains a central server which controls all decisions made by the network devices.

B. Conflict analysis

In literature, conflict analysis is mainly applied only to single types of security controls and there is no complete solution that incorporates all types of security controls. Research is mainly concentrated on Intra- and Inter-policy analysis of packet filter and IPsec configurations.

The conflict analysis of policy was first introduced by Al-Shaer and Hamed [3]. They presented a classification scheme for packet filter rule relations, based on which they defined the four types of intra-policy rule conflicts (shadowing, correlation, generalization and redundancy). Two rules are shadowed when they enforce different actions and both rules match the same packets. Two rules are correlated when they enforce different actions and both rules have some matching packets in common. A rule is a generalization of a second rule when they enforce different actions and the second rule matches the same packets as the first one but not vice versa. Two rules are redundant when they enforce the same action and match the same packets.

Al-Shaer et al. introduced an extension of the intra-policy classification analysis, called inter-policy rule conflicts, in the extension [10][11] of the first paper. Inter-policy analysis evaluates rule relations between serially-connected packet filters. Al-Shaer et al. define five new intra-policy conflicts (shadowing, spuriousness, redundancy, correlation and irrelevance). Two rules from two different firewalls are shadowed when they match the same packets and the rule from the first firewall blocks a packet that is permitted by the second rule. Two rules from two different firewalls are spurious when they match the same packets and the rule from the first firewall permits the packet which is blocked by the second rule. Two rules from two different firewalls are redundant when they match the same packets and both rules block the packet. Two rules from two different firewalls are correlated when they have some matching packets in common and enforce different actions. A rule is classified as irrelevant if there is no possible traffic which can be matched by the rule, for example the source and destination address belong to the same zone.

Based on the work of Al-Shaer et al., other researchers proposed alternative models and classification schemas. These works prove that Al-Shaers classification scheme is valid and can be applied to real world scenarios. The main limitation of all these approaches is that they cannot handle other security controls but packet filters. Notable examples are: Firecrocodile [12] and FIREMAN [13]. Firecrocodile [12], proposed by Lehmann et al., was the first approach to help network administrators to correctly configure PIX firewalls. The tool builds a model which represents the PIX configuration file and performs the analysis on it. In addition to conflict analysis they verify also the configuration file for policy violations. Its main limitation is that it can analyse only intra-policy packet filtering rules of Cisco PIX configurations. FIREMAN [13], proposed by Yuan et al., uses binary decision diagrams (BDDs) to represent packet filtering policies. In addition to a intra-policy analysis, it also verifies that an end-to-end policy is correctly implement by the filtering configurations. The model is designed for packet filters only and does not support any other type of security control.

Garcia-Alfaro et al. [14] propose the integration of network intrusion detection systems (NIDS). The model can detect both intra- and inter-policy packet filter rule conflicts. The main improvement over Al-Shaer's model is that it can also handle NIDS, and not only packet filters. The tool can also verify which security controls are on the path of a given packet based on its source and destination address. Another feature of this model is that it can rewrite a policy in its positive or negative form. The positive form of a policy contains only ALLOW rules whereas the negative form contains only DENY rules. This work has been later integrated into the MIRAGE tool [8].

Abbes et al. [15] suggest a different approach to this topic by using an inference system to detect intra-policy conflicts. They use the inference system to construct a tree representation of the policy. The construction process is efficient and optimized for memory consumption. The inference contains a condition which stops the construction of a specific branch when no conflict can be found. The resulting classification tree contains potential rule conflicts in its leaves. The disadvantage of this approach is that it is not able to check for inter-policy conflicts, furthermore it is not capable of handling security policies such as IPsec/VPN.

Only recently stateful firewalls have been integrated into analysis models. One of the few examples is presented in [16] and [17]. Cuppens and Garcia-Alfaro [16] propose a solution for intra-policy analysis of stateful firewalls. With the introduction of stateful firewalls they also present new types of conflicts classes (intra-state and inter-state rule conflicts). Intra-state rule conflicts occur only between stateful rules and beside the known conflicts from the stateless analysis, they include two new conflict types. The first new conflict arises when the firewall blocks packets during the three-way

 ISBN: 978-1-61208-378-0

handshake. The second new conflict arises when the firewall blocks packets during the connection termination. Inter-state rule conflicts occur between stateful and stateless rules when application layer protocols establish multiple connections and at least one of this connections is blocked, an example of such a protocol is FTP. The proposed algorithmic solution to handle and eliminate such types of conflicts is based on a general automata describing the stateful rules. This initial work has been completed and formalized in [17]. Although the introduction of stateful firewall into the analysis process was a very important step, both solutions are still missing the inter-policy analysis.

Basile et al. [18] present an new analysis model based on the work of Al-Shaer. The authors introduce a new formal model for policy specification, named Geometrical Model, it is based on a set of rules, a default action and an ad hoc resolution strategy. The presented model can identify all types of intra-policy conflicts defined by Al-Shear. Furthermore, the authors present two new conflict types: general redundancy anomaly and the general shadowing anomaly. The general redundancy anomaly occurs when a rule is redundant to the union of multiple rules. The general shadowing anomaly occurs when a rule is shadowed by the union of multiple rules.

Basile et al. [19] present, based on their Geometrical Model, a extension which can perform conflict analysis of application-level firewall configurations. The extended model can identify all policy anomalies introduced in their previous work. The main contribution of this work is the conflict analysis of firewall rules including regular expressions. The model transforms the regular expressions into deterministic automata and calculates rule intersection based on them.

Fu et al. [20] present a first approach for IPsec policy conflict detection. The analysis is performed on a set of policy implementations written in a high-level language and the policy conflicts are identified by verifying the implemented policies against a desired one. Fu et al. define a conflict when the policy implementations do not satisfy the requirements of the desired policy. A simple example of such a policy conflict is when the desired policy specifies that node A must have an encrypted channel with host B, but the policy implementations do not instantiate an encrypted channel from A to B. In addition to conflict detection, the proposed solutions includes also conflict resolution. The conflict resolution process tries to find alternative policy implementations in order to satisfy the desired policy.

Al-Shaer [21] formalizes the classification scheme of [20]. The proposed model not only incorporates the encryption capabilities of IPsec, but also its packet filter capabilities. The work can be seen as the extension of its packet filter classification proposed by Al-Shaer et al. [11]. In particular he identified two new IPsec conflicts (overlapping-session and multi-transform conflict), both types are valid for inter and intra-policy analysis. Nested session conflicts occur when multiple IPsec session are established from the same source to different remote hosts and the traffic is delivered to the farther host before the nearer one. Multi-transform conflicts occur when traffic protection is applied to already encapsulated IPsec traffic and the second protection is weaker than the first one. Al-Shaer presents in [22] a complete taxonomy of policy conflicts concerning packet-filter and IPsec configurations. This is the only approach who tries to perform conflict analysis of two different security controls.

Li et al. [23] present a similar detection classification model for IPsec security policy conflicts. The model takes in consideration intra- and inter-policy conflicts but is not compatible with the packet filter rule classification model presented by Al-Shaer. Instead of the conflicts defined by Al-Shaer they present a new alternative one. The new classification scheme is essentially the same but has the advantage that its definition is clearer and therefore easier to implement.

Niksefat and Sabaei [24] present a improved version of Al-Shaer's [21] solution. The new detection algorithm can identify all IPsec conflicts defined by Al-Shaer but does not support filtering conflicts. The solution uses a Binary Decision Diagram (BDD) to represent IPsec policies. The main improvement over Al-Shaer's solutions is the performance of the implementation. Beside the improved efficiency in the implementation this approach can also resolve the detected conflicts.

C. Reachability analysis

Reachability analysis can be performed both online and offline. Online reachability analysis is performed on a deployed system by injecting test packets and verifying on different points of the network that those packets are present. Offline reachability analysis is performed on a model of the system without direct interaction with a real network.

Online reachability analysis in general is performed by using tools such as *ping*, *traceroute*, and *tcpdump*. There are only a few publications regarding this topic. The general approach taken in literature is to insert a traffic generator and a traffic analyser into the network. The most promising work is presented by El-Atawy et al. [25] and Al-Shaer et al. [26], they propose a traffic generator which analyses first the security policy and based on this analysis, the most relevant packets are generated. The limitation of this two approaches is that they can be applied to single firewalls only.

Offline reachability analysis has the advantage that the system to be analysed does not need to be deployed. This means that it can be used during the design and maintenance tasks. Furthermore, it can also verify reachability on alternative paths, and therefore test fault-tolerance properties of the systems.

Mayer, Wool, and Ziskind [27] present a firewall analysis engine called Fang. It is the first approach towards offline reachability analysis of computer networks containing only packet filters. The proposed solution takes as input the network topology and the configuration files of the deployed packet filters. A user interface to perform reachability queries is provided and the queries are evaluated by the tool. In the extended versions of the paper [28] and [29] the query interface has been improved and the most relevant queries are generated automatically by the tool.

Xie et al. based there reachability analysis on graph theory and dynamic programming [30]. The solution is able to calculate the upper and lower bound of reachability. The upper bound defines that there is at least one possible path for reachability and the lower bound defines that all possible paths allow reachability. The model can be used to represent

 ISBN: 978-1-61208-378-0

static NAT, routing and filtering rules based on the destination addresses, but it does not take into account the existence of connectionless and connection-oriented protocols. Although the correctness of the model is given, it is purely theoretical and lacks experimental results. Bandhakavi et al. [31] present an extension to Xie's work to overcome limitations. They use a more general model to describe firewalls, packet filtering and transformation rules, thus adding the possibility to handle policies that depend on source addresses and filtering states.

Khakpour and Liu [4] present a reachability analysis tool called Quarnet. Quarnet supports connectionless (stateless router/firewall and static NAPT) and connection-oriented transport protocols (stateful router/firewall and dynamic NAPT). The paper presents a model for calculating network reachability metrics and also includes a performance analysis. The solution is based on an internal representation of the network on which reachability queries are executed. The authors first calculate a Firewall Decision Diagram (FDD) to represent the global policy and afterwards compute two matrices which contain the effective reachability information needed. Although the single reachability queries are very fast to compute, it takes quite a long time to compute the internal representation of the network.

Another theoretical approach used to compute the network-wide reachability, has been proposed by Sveda et al. [32]. This approach uses traditional graph-based algorithms, such as Floyd-Marshall, whereas [30] and [31] require ad-hoc techniques to mimic routing protocols. To calculate the reachability of the network the authors use the encoding problem into SAT instance solved by automatized solvers. They describe how to represent both routing and filtering devices, but do not mention how to express packet transformation rules.

Kazemian et al. [33] present a generalization of Xie's work [30] based on "Header space" information of packets. Their algorithm is compatible with filtering, routing, and transformation technologies. However, this approach is limited to packet filters and cannot be used for filtering and security devices which work at a higher level of the ISO/OSI stack.

D. Policy comparison

Fu et al. [20] present a solution proposal to verify the correct implementation of IPsec policies. The algorithm presented takes as input high-level security policies describing an implementation and compares it with a desired end-to-end policy. Even though the algorithm is able to compare a desired policy with its implementation, it cannot been used to compare a modified policy with its original version. Furthermore, it only supports IPsec policies and does not support routing or other transformation policies. This approach is more directed towards conflict analysis then policy comparison.

Liu et al. [34] and [35] propose to reduce configuration errors by forcing network administrators to write two separate concrete configurations and to compare them afterwards. The two configurations are converted into two FDDs and the comparison is performed onto the two FDDs. The comparison algorithm merges the two FDDs and verifies that the action, contained in the leaves of the tree, is the same at each point. Possible conflicts found in the two FDDs must be corrected manually by the administrators and without any correlation to the original configurations. This approach can be generalized and the two input policies may be seen as the original and the modified policy.

Yin and Bhuvaneswaran [36] represent correlations between rules as spatial relations and show how this special relations can be used to evaluate the impact of rule changes on the policy. Filtering policies are represented by the so-called SIERRA tree. A SIERRA tree is similar to a FDD, each level of the tree represents a dimension of the special division. The impact analysis can only be performed on single changes, such as adding one rule, removing or replacing it. The performance of the algorithm is very poor since to calculate the difference between two policies containing 30 rules takes already several seconds.

Liu et al. have published two papers on change-impact analysis of firewall policies [37][38], his algorithm is based on a FDD and supports the classic 5-tuple filtering rules. Overlapping rules are eliminated during the creation of the FDD and as a result the FDD represents a filtering policy without overlapping rules. The algorithm is designed to support four basic operations on firewall policies: rule deletion, rule insertion, rule modification, and rule swap. The output of the algorithm presents an accurate impact of a proposed change. Furthermore, the algorithm is also capable of correlating the impact of a policy change with a high-level security requirement. Although the authors claim that the algorithm is practical, neither of the two papers does a performance evaluation of the presented algorithms.

Liu et al. [39] present a firewall verification tool which takes as input a firewall policy and a given property. The tool verifies that the policy satisfies the given property. The tool is mainly useful for offline firewall debugging and troubleshooting. The algorithm first converts the firewall policy into a FDD and the verification process is performed on the FDD. The verification process checks that all leafs, which are correlated to the given property, enforce the desired action. A implementation of the tool has been tested for performance and shows excellent results. This solution is limited to one single firewall and cannot verify the correct implementation of a complete network.

Youssef et al. [40] propose a formal and automatic verification method based on a inference system. The solutions certify that a firewall configuration is sound and respect completely to a security policy. In case that the configuration is not sound and complete, the method provide the user with information tosolve the issues. This paper only supports packet filter firewalls; however in an extended version [41], Youssef et al. propose a formal and automatic method to check also statefull firewall configurations.

E. Summary

Table I summarizes the capabilities of the different approaches. Each row stands for one approach identified by its citation number. The three analysis categories (conflict analysis, reachability analysis and policy comparison) are separated by horizontal lines. Each column stands for a specific capability; the first four columns identifies the type of analysis (intra-policy conflict analysis, inter-Policy conflict analysis, reachability analysis, and policy comparison) and the last

 ISBN: 978-1-61208-378-0

seven columns identify the supported security control (packet filter firewall, stateful firewall, application-level firewall, NIDS, IPsec/VPN, NAT/NAPT, and routing).

TABLE I. SUMMARY

	Intra-Policy	Inter-Policy	Reachability	Comparison	Packet Filter	Stateful FW	Application	NIDS	IPsec/VPN	NAT/NAPT	Routing
[3]	⊗				⊗						
[10] [11]	⊗	⊗			⊗						
[12]	⊗				⊗						
[13]	⊗	⊗			⊗						
[14]	⊗	⊗			⊗			⊗			
[15]	⊗				⊗						
[16] [17]	⊗				⊗	⊗					
[18]	⊗				⊗						
[19]	⊗				⊗		⊗				
[20]	⊗			⊗					⊗		
[21] [22]	⊗	⊗			⊗				⊗		
[23]	⊗	⊗			⊗				⊗		
[24]	⊗	⊗							⊗		
[4]			⊗		⊗	⊗				⊗	
[27]			⊗		⊗						
[28] [29]			⊗		⊗						
[30]			⊗		⊗					⊗	⊗
[31]			⊗		⊗	⊗				⊗	
[32]			⊗		⊗					⊗	
[33]			⊗		⊗					⊗	⊗
[34] [35]				⊗	⊗						
[36]	⊗			⊗	⊗						
[37] [38]				⊗	⊗						
[39]				⊗	⊗						
[40]				⊗	⊗						
[41]				⊗	⊗	⊗					

By comparing the different approaches, the two major limitations are evident. Firstly, the majority of papers is concentrated only on packet filters and ignore other security controls. Secondly, the papers mainly focus only on one of the three analysis types (conflict analysis, reachability analysis and policy comparison), and only a few try to combine different approaches into one single model.

V. FUTURE RESEARCH

As it becomes clear from the analysis of research carried out so far, there is a lack of interoperability among the various models. This has three major disadvantages. Firstly, a security control modelled for one research approach cannot be reused in another one. Secondly, the execution time spent to instantiate a model is repeated for each and every analysis performed on network security policies. Thirdly, it is nearly impossible to make a performance comparison of the different approaches since they use different test scenarios or do not present a performance evaluation at all.

By combining all the proposed analysis techniques into one single extensible model, all of these disadvantages are eliminated and a proper analysis framework is created for future research. Firstly, after a security control has been modelled, evaluated and implemented it can be used by all types of analysis techniques. Secondly, when a network administrator wants to perform different types of analysis, he has to insert the required information and instantiate the model just ones. Thirdly, by having just one model, new algorithms can be evaluated by comparing them directly to each other.

To accomplish this goal, the new model should have some distinctive features, such as well-defined input formats, a flexible structure, and extendible bindings. Furthermore, the new model may include tests-scenarios to evaluate the performance of new algorithms.

The new model has to take as input the network topology, and the network security policies written in different formats and for different security controls. For example, it could take as input the global network security policy written in a technology-independent formal language and the complete network structure with all its concrete configurations. The model can then perform a policy comparison between the two input formats and verify that the implementation follows the desired network security policy. As a further step, network administrators can verify reachability of critical components or perform a conflict analysis for better understanding.

The new model has to be flexible to accommodate all types of security controls and network topologies. In order to support all types of computer networks, the model should be able to compose different security controls in different order. Security controls should be modelled so that they are completely independent form network topology.

The new model has to be extensible for new types of security controls. In order to be prepared for future security controls, the model has to be able to include new ones without significant changes to the model itself.

VI. CONCLUSION

Need for better tools to support network administrators is evident, from the number of publications regarding this topic. Although publications are very promising, they are only at the beginning. The analysis of articles has shown that the research is concentrated on quite small sub-problems and there exists no global solution to the problem.

By combining all research approaches into one single model, the impact grows in two dimensions: first, the number of possible analysis types, and second, the number of supported security controls. This leads to a model that can perform different policy analysis and, at the same time, covers a wider range of security controls. This approach leads to two improvements: firstly, reduced research effort and secondly, reduced execution time.

The research effort is mainly reduced because security controls have to be modelled only once and afterwards they can be used for different policy analysis. The execution time is reduced mainly because the model is shared by various policy analysis and its creation has to be performed only once.

ACKNOWLEDGMENT

The research described in this paper is part of the SECURED project, co-funded by the European Commission (FP7 grant agreement no. 611458).

REFERENCES

[1] A. Wool, “Firewall Configuration Errors Revisited,” CoRR, vol. abs/0911.1240, 2009, pp. 103–122.

[2] W. Avishai, “Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese,” IEEE Internet Computing, vol. 14, no. 4, July 2010, pp. 58–65.

 ISBN: 978-1-61208-378-0

[3] E. Al-Shaer and H. Hamed, “Firewall Policy Advisor for anomaly discovery and rule editing,” in IFIP/IEEE 8th Int. Symposium on Integrated Network Management, Colorado Springs, CO, March 24–28 2003, pp. 17–30.

[4] A. Khakpour and A. Liu, “Quarnet: A tool for quantifying static network reachability,” IEEE/ACM Trans. Netw., vol. 21, no. 2, February 2009, pp. 551 – 565.

[5] AlgoSec Inc., “Examining the dangers of complexity in network security environments: Algosec survey insights,” 2012.

[6] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato: A novel firewall management toolkit,” ACM Transactions on Computer Systems, vol. 22, no. 4, November 2004, pp. 381–420.

[7] P. Verma and A. Prakash, “FACE: A Firewall Analysis and Configuration Engine,” in 2005 Symposium on Applications and the Internet, Trento, Italy, January 31 – February 4 2005, pp. 74–81.

[8] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and P. Stere, “MIRAGE: a management tool for the analysis and deployment of network security policies,” in SETOP 2010: 3rd International Workshop, Athens, Greece, September 23 2011, pp. 203–215.

[9] M. Casado, T. Garfinkel, and A. Akella, “SANE: A protection architecture for enterprise networks,” in USENIX-SS06: USENIX Security Symposium, Vancouver, Canada, July 31 – August 4 2006, pp. 137–151.

[10] E. Al-Shaer and H. Hamed, “Discovery of policy anomalies in distributed firewalls,” in INFOCOM 2004, Hong Kong, Cina, March 7–11 2004, pp. 2605–2616.

[11] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” IEEE Journal on Selected Areas in Communications, vol. 23, no. 10, October 2005, pp. 2069–2084.

[12] N. Lehmann, R. Schwarz, and J. Keller, “FIRECROCODILE: A Checker for Static Firewall Configurations,” in SAM06: International Conference on Security & Management, Las Vegas, NV, June 26–29 2006, pp. 193–199.

[13] L. Yuan, H. Chen, J. Mai, and C.-n. Chuah, “FIREMAN: A Toolkit for FIREwall Modeling and ANalysis,” in IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, May 21–24 2006, pp. 199–213.

[14] J. Garcia-Alfaro, N. Boulahia-Cuppens, and F. Cuppens, “Complete analysis of configuration rules to guarantee reliable network security policies,” International Journal of Information Security, vol. 7, no. 2, April 2007, pp. 103–122.

[15] T. Abbes, A. Bouhoula, and M. Rusinowitch, “An inference system for detecting firewall filtering rules anomalies,” in SAC08: ACM symposium on Applied computing, Fortaleza, Brazil, March 16–20 2008, pp. 2122–2128.

[16] F. Cuppens, “Handling Stateful Firewall Anomalies,” in SEC2012: Information Security and Privacy Conference, Heraklion, Greece, June 4–6 2012, pp. 174–186.

[17] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot, “Management of stateful firewall misconfiguration,” Computers & Security, vol. 39, no. A, November 2013, pp. 64–85.

[18] C. Basile, A. Cappadonia, and A. Lioy, “Network-Level Access Control Policy Analysis and Transformation,” IEEE/ACM Transactions on Networking, vol. 20, no. 4, August 2012, pp. 985–998.

[19] C. Basile and A. Lioy, “Analysis of Application-Layer Filtering Policies With Application to HTTP,” IEEE/ACM Transactions on Networking, vol. PP, no. 99, December 2013, pp. 1–1.

[20] Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu, “IPSec/VPN Security Policy: Correctness, Conflict Detection, and Resolution,” in International Workshop, POLICY 2001, Bristol, UK, January 29–31 2001, pp. 39–56.

[21] E. Al-Shaer, H. Hamed, and W. Marrero, “Modeling and Verification of IPSec and VPN Security Policies,” in 13th IEEE Int. Conference on Network Protocols, Boston, MA, November 6–9 2005, pp. 259–278.

[22] E. Al-Shaer and H. Hamed, “Taxonomy of conflicts in network security policies,” IEEE Communications Magazine, vol. 44, no. 3, March 2006, pp. 134–141.

[23] Z. Li, X. Cui, and L. Chen, “Analysis And Classification of IPSec Security Policy Conflicts,” in FCST06: Japan-China Joint Workshop on Frontier of Computer Science and Technology, Fukushimna, Japan, November 17–18 2006, pp. 83–88.

[24] S. Niksefat and M. Sabaei, “Efficient Algorithms for Dynamic Detection and Resolution of IPSec/VPN Security Policy Conflicts,” in 2010 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, WA, April 20–23 2010, pp. 737–744.

[25] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li, “An Automated Framework for Validating Firewall Policy Enforcement,” in POLICY07 : 8th IEEE International Workshop on Policies for Distributed Systems and Networks, Bologna, Italy, June 13–15 2007, pp. 151–160.

[26] E. Al-Shaer, A. El-Atawy, and T. Samak, “Automated pseudo-live testing of firewall configuration enforcement,” IEEE Journal on Selected Areas in Communications, vol. 27, no. 3, April 2009, pp. 302–314.

[27] a. Mayer, a. Wool, and E. Ziskind, “Fang: a firewall analysis engine,” in 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, May 14–17 2000, pp. 177–187.

[28] A. Wool, “Architecting the Lumeta Firewall Analyzer,” in 10th USENIX Security Symposium, Washington, DC, August 13–17 2001, pp. 85–97.

[29] A. Mayer, A. Wool, and E. Ziskind, “Offline firewall analysis,” International Journal of Information Security, vol. 5, no. 3, July 2006, pp. 125–144.

[30] G. Xie, D. Maltz, A. Greenberg, G. Hjalmtysson, and J. Rexford, “On static reachability analysis of IP networks,” in INFOCOM2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, FL, March 13–17 2005, pp. 2170–2183.

[31] S. Bandhakavi, S. Bhatt, C. Okita, and P. Rao, “Analyzing end-to-end network reachability,” in IM09: IFIP/IEEE Int. Symposium on Integrated Network Management, Long Island, NY, June 1–5 2009, pp. 585–590.

[32] M. Sveda, O. Rysavy, and G. D. Silva, “Static Analysis of Routing and Firewall Policy Configurations,” in ICETE10: 7th International Joint Conference, Athens, Greece, 2012, pp. 39–53.

[33] P. Kazemian, G. Varghese, and N. McKeown, “Header space analysis: Static checking for networks,” in NSDI12: 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA, April 25–27 2012, pp. 9–9.

[34] A. Liu and M. Gouda, “Diverse Firewall Design,” in Int. Conference on Dependable Systems and Networks, Florence, Italy, June 28 – July 1 2004, pp. 595–604.

[35] A. Liu and M. Gouda, “Diverse Firewall Design,” IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 9, September 2008, pp. 1237–1251.

[36] Y. Yin and R. Bhuvaneswaran, “Inferring the Impact of Firewall Policy Changes by Analyzing Spatial Relations between Packet Filters,” in ICCT06: Int. Conference on Communication Technology, Guilin, China, November 27–30 2006, pp. 1–6.

[37] A. Liu, “Change-impact analysis of firewall policies,” in 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24–26 2007, pp. 155–170.

[38] A. Liu, “Firewall policy change-impact analysis,” ACM Transactions on Internet Technology, vol. 11, no. 4, March 2012, pp. 1–24.

[39] A. Liu, “Formal Verification of Firewall Policies,” in 2008 IEEE International Conference on Communications, Beijing, China, May 19–23 2008, pp. 1494–1498.

[40] N. Ben Youssef, A. Bouhoula, and F. Jacquemard, “Automatic verification of conformance of firewall configurations to security policies,” in 2009 IEEE Symposium on Computers and Communications, Sousse, Tunisia, July 5–8 2009, pp. 526–531.

[41] N. B. Youssef and A. Bouhoula, “Dealing with Stateful Firewall Checking,” in DICTAP2011, Dijon, France, June 21–23 2011, pp. 493–507.

 ISBN: 978-1-61208-378-0

Notification Support Infrastructure for Self-Adapting Composite Services

Erlend Andreas Gjære, Per Håkon Meland and Thomas Vilarinho
SINTEF ICT, Software Engineering, Safety and Security
Trondheim, Norway
Email: {erlendandreas.gjare, per.h.meland, thomas.vilarinho}@sintef.no

***Abstract*—Building reliable software services based on service components supplied by partners and third parties in potentially complex chains of providers, is inherently challenging. In the case of cloud-based services, providers may offer not only software (web services) but also infrastructure (e.g., processing and storage). Making composite services trustworthy and reliable requires all parties to be open about changes that may impact their customers, and they must inevitably deal with a fluctuating threat picture. In this paper, we describe a publish/subscribe-based notification infrastructure which allows a Service Runtime Environment (SRE) to receive alerts about changes and threats in service components. Notifications arise from both human and automated monitoring, and are published to a notification broker which handles subscriptions and message distribution. The SRE is enabled to react to these notifications automatically through adaptation of the service composition, based on rules that can syntactically match the contents of received notifications. In addition to describing the technical implementation, we show an example of how a composite service from the Air Traffic Management (ATM) domain can instantly adapt itself when it receives a notification about a threatened service component. We also demonstrate how a mobile client app is used to keep humans aware of the notifications.**

Keywords–Security notifications; self-adaptability; accountability, composite services.

I. INTRODUCTION

With the emerging paradigm of composite services, i.e., software services composed of functionality provided by several services components—changes or attacks on a service component implies an attack or change to the composite service as a whole. Service compositions in general are highly distributed and have a complex nature. They expose a greater attack surface than traditional stand-alone systems, and introduce multiple layers of policies, which may not be compatible. Web services and cloud-based software components may be offered by third party parties and potentially in long provider chains, and can be used in compositions along with other services. In the age of cloud computing, the importance of breach notifications is emphasised due to the need for transparency throughout entire chains of service providers—seeking trustworthiness through accountability [1].

To make these kinds of services reliable enough, we need a holistic approach to how they are built in terms of security and accountability. The entire line-up of involved service components and providers must be considered in terms of how situations need to be mediated and mitigated where (unwanted) changes or threats occur [2]. Simply having control over which relevant third parties (and fourth parties, etc.) our provider deals with is of course a place to start. Preventive measures in this context today are to a large extent being based on risk assessments, contractual relationships and audits. Reactive measures, on the other hand, include notifications via email, phone, remote log file-streams and dashboards for mediating significant changes and security incidents. The contents of these are however not as predictable as needed for triggering corrective activities *automatically* in composite services. Preparing for "failure" is equally needed to prepare a composite service for reacting this way.

To be able to respond to changes in an effective and truly timely manner, we need notifications that are machine-readable and syntactically clear in such a way that they can be used to trigger automatic adaptation of a composite service. This, however, requires a higher level of refinement than traditional log streams provide, and either automated or human processing of low-level input may be needed in advance to infer (and mediate) useful correlations between several more atomic log events. At the same time, notifications being sent if only a change or threat is actually encountered, could allow organisations to exchange more information without revealing "too much". We therefore seek a notification infrastructure where service providers report in a convenient and standardised way on behalf of themselves, and are equally able to receive notifications from the others corresponding to which service components are being used at any time.

There are strong regulatory reasons why such a notification infrastructure needs to be realised. The European Union implemented a breach notification law in the Directive on Privacy and Electronic Communications (ePrivacy Directive) in 2009 [3]. In this directive, it is stated under article 4, paragraph 2 that *"In case of a particular risk of a breach of the security of the network, the provider of a publicly available electronic communications service must inform the subscribers concerning such risk and, where the risk lies outside the scope of the measures to be taken by the service provider, of any possible remedies, including an indication of the likely costs involved."* Similarly, security breach notification laws have been enacted in most U.S. states since 2002 (the only states with no law are currently Alabama, New Mexico and South Dakota) [4].

Our main goal with this paper is to describe a notification infrastructure that can support automatic application of corrective measures to service compositions at runtime. The approach includes a publish-subscribe based infrastructure for message delivery, along with a way to prepare rules at design-time for enabling automated response at runtime. We demonstrate our

 ISBN: 978-1-61208-378-0

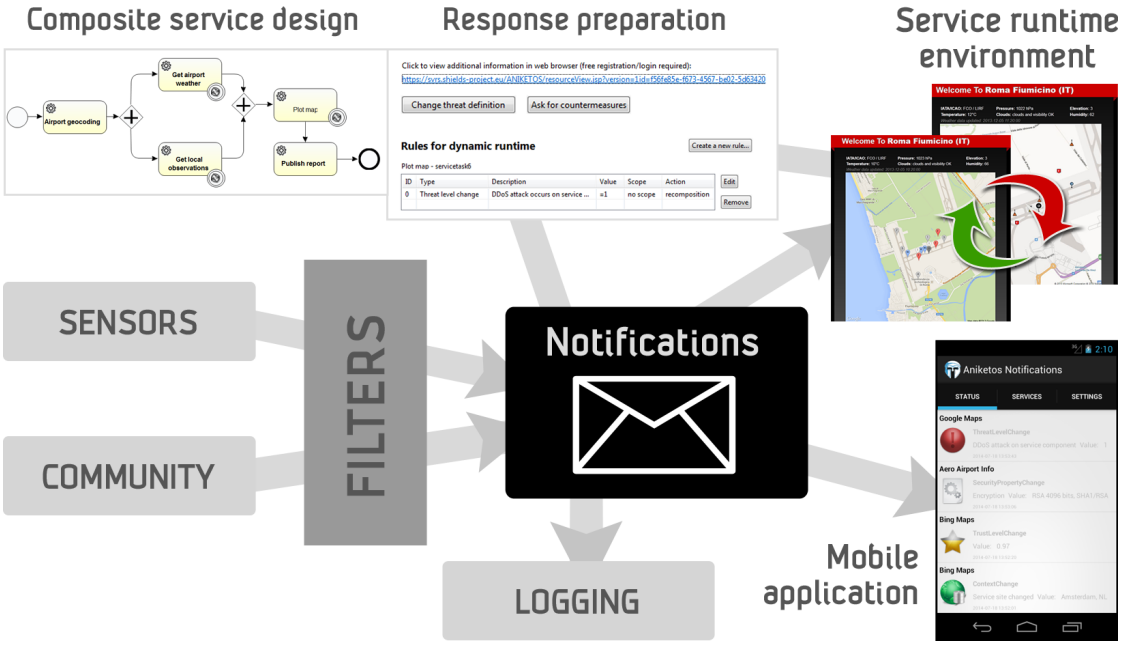

Figure 1: An outline of how composite services can be prepared and deployed ready for automatic adaptation, supported by an infrastructure for security notifications.

infrastructure by allowing a composite service in our Service Runtime Environment (SRE) to dynamically substitute one of its components without downtime, upon receiving notification about a threat. We also describe our accompanying mobile client app, which receives the same notification and can show them to a layperson, not necessarily a security expert, but for instance someone who has some kind of responsibility for the composite service.

Our work complements similar approaches from the past, such as Zheng and Lyu [5], who propose a user-collaborative failure data sharing mechanism for service-oriented systems. What separates our work from the rest is the focus on security violations and threats, as opposed to reliability in general.

This paper is structured as follows: Section II presents our approach at a high level including a user scenario that describes how to prepare for escalations, and what the output might be. Technical details of our implementation work are provided in Section III. Section IV discusses various technical design choices, shortcomings and challenges, related and future work, before Section V concludes the paper.

II. Approach

Figure 1 shows how our notification infrastructure can be used to support automatic adaptation of composite services. In this section, we illustrate the approach with a practical use case example from the Air Traffic Management (ATM) domain: The pilot of an aircraft uses our composite service to retrieve up-to-date information about the airport to where the flight is destined. The airport report is built from data pulled simultaneously from several web services and service providers. It includes both geographical information, the current weather, a few local observations related to, e.g., oil or water on the runway, and finally, a map onto which this information is plotted.

A. Notifications

At the core we have the notification messages, which tie everything together. These are designed in particular to facilitate real-time communication of security and change events within distributed systems (of systems), especially between service providers and across geographical boundaries. We do not claim it to be a perfect design—it is a prototype—but we do have a version, which supports automated adaptation of composite services based on a number of security requirements [6].

Each notification message needs to identify the resource it concerns. Since composite services deal with web services, and all such services each have their own public endpoint Uniform Resource Locator (URL), we use this URL as the unique service ID. The service ID is needed to create subscriptions and to match received notifications with local rules in the SRE. The notifications are further typed into a specific category, i.e., one of those specified in Table I (first column). Some of the notifications types have support for a few sub-types in a hierarchy below. These sub-types may be essential to provide an appropriate response to the received notification, for instance where different sub-types of threats require different countermeasures, and commonly agreed values here are needed. There is also a value field required for each notification. The value is needed to specify the actual status of the notified type (and sub-type), e.g., if a threat is on the rise or if it has passed, which is expressed with probability as a float value between 0 and 1. The value can also be a simple string, or even a Javascript Object Notation (JSON) [7] object (a human-readable format for transmitting data). Self-adaptation may be based on the value, so there is a need to

 ISBN: 978-1-61208-378-0

TABLE I: NOTIFICATION TYPES AND CONTENT SPECIFICATION

ThreatLevelChange	Supports several sub-types of threat families, e.g., *DDoS attack*, *Service injection*, *Vulnerable crypto library*, *Trust poisoning*, etc. Value specifies the probability of a threat being encountered (between 0 and 1). Also possible to specify a threat ID from a common threat repository for linking to a very specific threat (and more info on-line).
SecurityPropertyChange	Supports sub-types with various security properties to detail how a service is implemented in terms of security, e.g., *Encryption*, *Backup cycle*, etc. Implementation details are to be provided in the value field, such as length of backup cycle.
ContextChange	Supports sub-types describing the context of the service, e.g., *Server site location*, *Backup location*, etc. In these particular cases, a location name (e.g., country) would go into the value field.
ContractViolation	Supports sub-types for violated properties in cases where a violation is detected during service contract matching with consumer policy, e.g., *Separation of duty*, *Binding of duty*, etc.
ServiceChange	Intended for composite services to provide an update if a recomposition has been made and hence a service component has been substituted. Sub-types such as *Component added* and *Component removed* can be used, or even *Service not provided* if a service is taken (temporarily) down due to a vulnerable component that could not be replaced.
TrustLevelChange	Specifies a level of trustworthiness (between 0 and 1). The values need to be published by an authoritative source for trust ratings (not self-declared).

standardise a format to use for each individual notification type (and sub-type) here.

B. Composite Service Design and Response Preparation

To create composite services we use the Business Process Model and Notation (BPMN) [8] and a modified graphical tool [6] to model the service specification as a process consisting of BPMN service tasks. In our example shown in Figure 2, visible as boundary error intermediate events [9], we have modelled a threat *DDoS-attack occurs on service component* due to the importance of high availability. The Distributed Denial of Service (DDoS) threat can in this case be both monitored and dealt with quite easily, so we want to be notified whenever an attack occurs. Instead of just allowing the attacked component to bring down our entire composition, we need to prepare the composite service to switch automatically to another map component not under attack.

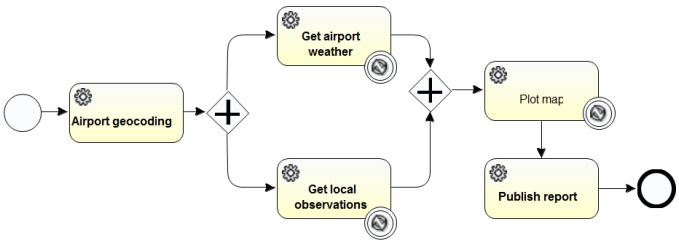

Figure 2: A composite service process is modelled in BPMN, with triggering events for notification on changes and threats.

Based on the triggering events modelled in the composite service definition, rules can be defined in the model to address the scenarios we are able to foresee. It is the service tasks, not the actual web service components, which are targeted in the scope of these rules. This makes the rules independent of the actual service component implementations we choose to deploy, and we do not need to define individual rule-sets for each potential composition plan.

For each rule, one must define the same properties as a notification would be expected to carry to get a match. This includes setting the notification type, sub-type and/or a value with some comparison of either equal to (also used for string comparison), larger than (or equal) and smaller than (or equal). In the scope section of the rule editor, it is further possible to define where in the process the rule shall apply. If a particular threat or change occurs for a component, we might not need to perform reactive measures unless it happens before, after or during the execution of a particular task in our process. We also have an option to launch an additional service process, e.g., one that might initiate hardening of other parts of the system, and/or send notification messages to clients and/or service technicians. In the end, we might end up with a list of several rules for several service tasks, or simply one for the one we have defined in our case study, as shown in Figure 3. Recent work by Salnitri et al. [10] goes into more details on rule definition and execution.

To perform a recomposition in our use case, we need an alternative service specification ready where another functionally equal service component is used to realise the map plotting task. We define one composition plan where the report generation is done based on the map service from Google, and we define another where the same responsibility is served by Bing. One of these plans will be then deployed as default. When there is a threat notification and the other composition plan is not affected by it, then this plan is considered more appropriate and can be deployed immediately in place of the attacked one. It is, however, not always the case that we have two candidate components for the same service task. One might also set the rule to simply stop providing the entire composite service, e.g., if a non-replaceable component has changed its policy and becomes no longer compliant.

C. Filtered Input From Sensors and Community

In order to generate notifications in the first place, there must obviously be some kind of monitoring in place for service components. In the case of cyber-threats, some service providers have dedicated security operations to monitor and process low-level input from e.g., syslog. Security Information and Event Management (SIEM) systems are used to gather and correlate security events from multiple logging sources both in real time and aggregated from previous events, and are sometimes purchased as a service from third party providers. This, however, does not normally involve logs from other organisations, except when changes are already publicly disclosed. Some monitoring efforts are also performed on a national level or sometimes by joint efforts across, e.g., an entire business sector, such as the Norwegian financial sector cybercrime unit [11]. Such intelligence could be a valuable source of input for an even wider community in the global service market. Apart from trustworthiness ratings, which need to come from an independent source, it should at least be possible to publish alerts for services provided by oneself.

 ISBN: 978-1-61208-378-0

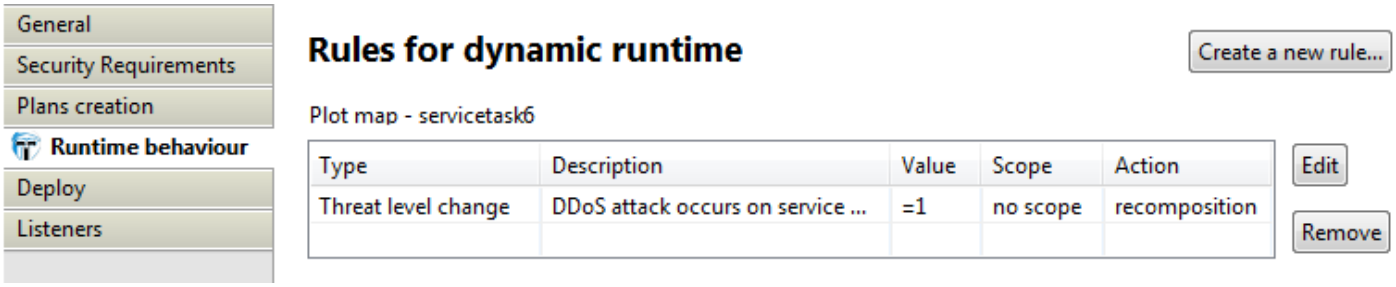

Figure 3: A rule has been defined for responding to a DDoS-attack on the map service.

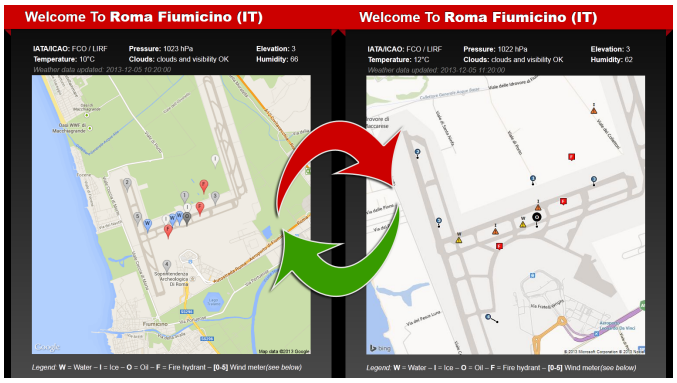

Figure 4: The SRE has replaced the Google map service with Bing Maps through a recomposition.

In the security community, there are also people who for instance create scanner bots, which are used to check the Internet for particular vulnerabilities, such as in the case of the *Heartbleed* bug [12]. A threat "*Vulnerable crypto library*" could then be warned against with the notification infrastructure, both when it is publicly disclosed and detected on a particular server, as well as immediately when the server has been patched (with a 0 value). In the particular case of Heartbleed, it would also be possible to add a threat ID referring to, e.g., the Common Vulnerability and Exposures (CVE) database [13] (in this case CVE-2014-0160), which would provide a pointer to more specific information. Assurance from the security community would however be necessary to validate the notifications, or else it would be easy to inject malicious notifications and cause severe business disruption for service providers who are shut out.

Regardless of input source, notifications are collected through the common web service interface provided for publishers. For SIEM or Security-as-a-Service (SaaS) products, it is easy to add support for invoking this notification service by an operator. We have implemented a web interface to be available for manual reporting, and have integrated other independent monitoring platforms, such as a threat monitoring module [14], which is able to infer the required level of semantics and trigger appropriate notifications. As long as the format can match what the SRE is able to interpret from the notifications, the rules are in practice agnostic to monitoring implementations.

D. Notification Receivers

In order to trigger events in the self-configuring service process, the SRE must be able to receive such notifications and align these with the previously defined and deployed rules. Since a threat monitor has now purportedly detected that the original map service used in our case study is hit by a DDoS-attack, the SRE is notified about this through subscriptions based on the deployed service composition. As it can find a match with the rule we previously defined on DDoS-attacks, the SRE initiates the specified action according to that rule, which is here to try a recomposition.

Since we had support for two different map services providing the same functionality, we have in practice prepared an additional composition plan for the airport report composite service. When the notification concerning a DDoS-attack on the map service is triggered and received by the SRE, a service verification mechanism [15] can tell us that the first plan no longer satisfies our security requirements. The rule in Figure 3 will initiate a recomposition accordingly. The original composition plan with the DDoS-ed map service will be ignored, and the second plan becomes the top-ranked. The recomposition proceeds with deploying the second composition plan, containing the alternative map service instead of the original one. Nevertheless, the same level of functionality is provided, as illustrated with Figure 4 where the airport reports, before (to the left) and after recomposition, are lined up next to each other.

While monitoring and responding to changes and threats in real time is our main goal, we also need to store the notifications for future reference. A common repository for historical security and service change notifications is valuable for doing research on previous events. There are of course questions to answer in this respect, e.g., how the repository should be managed and to what level of explicitness historical data should be made accessible over time. A repository could provide insights on questions like, e.g., cascading events between service providers, temporal aspects of incident/response, etc. Issues that must be dealt with in this context, as well as for the industry as a whole, is about the fear of negative publicity, customer repercussions and lost revenue caused by being "too" open. Instead of being seen as accountable, some may see service providers that do their job on reporting incidents as having inferior security.

 ISBN: 978-1-61208-378-0

TABLE II: MESSAGING PATTERNS APPLIED IN OUR IMPLEMENTATION

Event message	Notification messages are used to transmit events from one service to another. These notifications are reliable and asynchronous.
Publish-Subscribe Channel	The provider of the notification sends the event on a publish-subscribe channel, which delivers a copy of a particular event to each receiver. The receivers must have expressed an interest in such event on beforehand.
Data-Type Channel	The different notification types enables the receivers to easily interpret the incoming events.
Guaranteed Delivery	The notification infrastructure uses a store and forward mechanism to ensure message durability.
Message Bus	Independent systems and services of various types are able to communicate in a loosely coupled fashion through the use of the message bus pattern.
Event-Driven Consumer	The receving parties automatically consume messages as they become available.

III. Implementation Details

To operate a self-adapting distributed service infrastructure at runtime, a supporting system for Machine-to-Machine (M2M) messaging across networks was needed. Since there may be an endless number of services and SREs utilising this infrastructure, we cannot provide all messages to everyone. A publish-subscribe channel pattern was found suitable for this, since each service provider already needs to define which service components are to be used in a service composition. Appropriate subscriptions can be derived automatically from these service specifications. Registering the subscriptions should be handled by the SRE, which will then receive notifications only about threats for relevant services.

In addition to publish-subscribe, we have designed the notification infrastructure according to a number of messaging patterns in service-oriented architectures. Table II gives an overview of these, based on Chatterjee [16].

A. Notification Message Broker

The entire infrastructure relies on the delivery of notifications to subscribers. In a publish-subscribe architecture, this is the responsibility of the message broker. In addition to dispatching the messages and providing an endpoint for subscriptions, the broker will receive notifications to publish from authorised publishers.

Our implementation builds on Apache ActiveMQ [17] in this role, utilising the de-facto standard Java Message Service (JMS) [18] specification to provide compatibility with many platforms and messaging protocols. A broad variety of wire level protocols are supported, meaning that brokers can be connected to clients built with any programming language or platform with a compatibility for just one of these protocols. The performance of ActiveMQ brokers can be scaled up horizontally by configuring several instances in a network of brokers, if needed. Our broker is in addition deployed on a cloud-based infrastructure, for further increase in scalability.

The notifications are organized in a hierarchy of JMS topics and sub-topics, in practice building a single subscription string. The first part of the subscription string contains the service ID. Then, below that topic there are different sub-topics mapping to the different security notification types, each part of the subscription string separated with a dot '.'. With this hierarchy, one could decide to subscribe to all notifications from a service or particular types or sub-types within that service. If no type or sub-type is specified, the wild-card '>' will match all topics from that character and onwards in the subscription string. In this way, topics for each of the notification types can be created dynamically, without needing to explicitly subscribe to all of them individually, and without risking topics that no-one subscribes to. Hence, new sub-topics can be added and those subscribing to the main topic will start receiving notifications from the new sub-topic as well.

In addition, the entire hierarchy has common root nodes where one can set the access control level for all notifications below them. With that in mind, we created an entirely public channel ("pub") that anyone can subscribe to anonymously, but not publish through. For publishing notifications, the publishing client needs to authenticate to the broker. Apache ActiveMQ authentication/authorization configuration capabilities offer the possibility of implementing authentication able to access/subscribe to notifications from another channel, e.g., a service level that for instance would require formal contracts to be established between participating parties in advance. An ActiveMQ JMS topic corresponding to a service notification will finally have the format *pub.[serviceName].[notificationType].[subType]*. So, for example, the topic *pub.http://demo-aniketoswp6.rhcloud.com/googlemap/service.ThreatLevelChange.DDoS attack on service component* would map into:

- **Channel**: Public
- **Service ID**: http://demo-aniketoswp6.rhcloud.com/googlemap/service
- **Notification Type**: ThreatLevelChange
- **Sub-Type**: DDoS attack occurs on service component

In our implementation, we have granted subscription and notification retrieval access to all clients (anonymous access), and publish access only to authorized users. However, this set-up could be configured differently on the broker, allowing specific settings per topic or subtopic. For the authentication regarding the authorized access, we have used a simple XML configuration file, mainly because it was just a prototype implementation. In a real deployment, one could have relied on ActiveMQ support for Java Authentication and Authorization Service (JAAS) [19] or LDAP. When it comes to encryption of the messages, we have been using Transport Layer Security (TLS) encryption towards both publishers and subscribers. However, due to a limitation of the Android mobile client library used for the wire protocol, we had to support messaging in clear text towards the mobile client app.

B. Service Runtime Environment

The SRE is responsible for executing the composite service at runtime. We have implemented ours using the Activiti Engine [20], which can take BPMN composition plans as input, as they have been built with our graphical modelling tool. In the runtime engine, all BPMN is converted to executable code, based on the standard and without further manual work.

For the SRE, we have a plug-in [6] that is able to connect to the notification broker and create subscriptions

 ISBN: 978-1-61208-378-0

based on the rules attached to a service deployment. Whenever a composition plan is deployed, it includes all service IDs needed to create the subscriptions. For each service ID in the composition, there is a corresponding JMS topic being an individual channel of notifications, which a client can subscribe to. Accordingly, when subscribing to notifications for several services, there must be registered individual subscriptions for each of these services. The granularity of the rules will determine the relevance in cases of, e.g., slight variations in threat levels (not all changes will actually trigger an action).

The plug-in is a Java client implementation using the ActiveMQ library to connect to the broker. When subscriptions are made, the plug-in starts to listen for incoming notifications according to the made subscriptions. When a notification arrives, it compares it with all active rules for a potential match. The rules are simple XML-representations of what was described at design-time, having elements for each of the notification properties to match, as well as an element controlling scope and one controlling action(s) to perform.

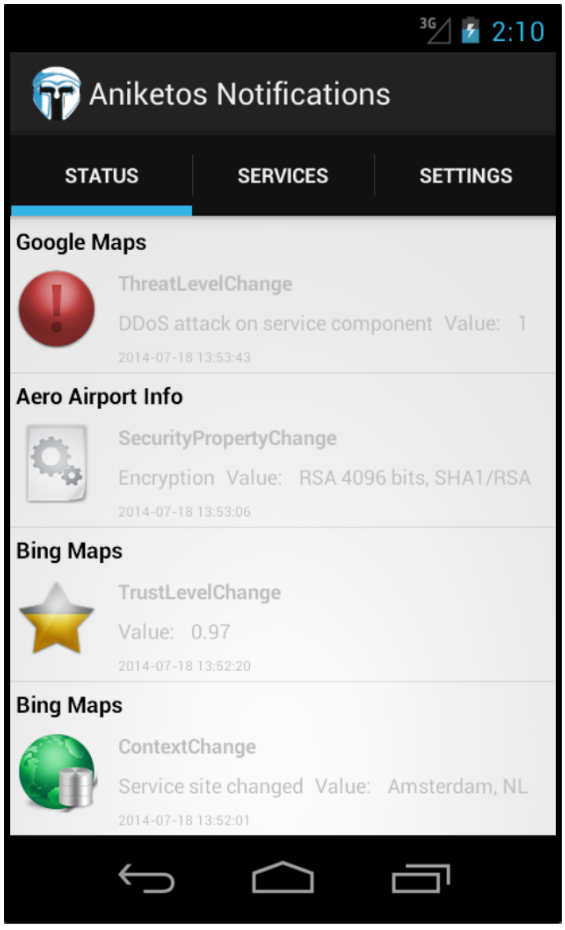

Figure 5: Screenshot of the Android client app for receiving notifications on-the-go.

C. Android Client App

The client app is perhaps not a critical part of the infrastructure, but might indeed be a useful one. While we have a reference implementation available for a regular Java-based subscription client—and any client can subscribe to the topics using any one of the protocols supported by the ActiveMQ broker—this does not fit very well with the usage scenario of receiving updates at *any* time. A mobile client app, on the other hand, can always be brought along by the person(s) responsible for monitoring a service, listening silently in the background and then notifying when something happens.

A working implementation of our client app for the Android mobile operating system has been made. Its functionality is currently rather simple, just enabling users to subscribe (and unsubscribe) to services, and show a feed of the updates. The app does not support choosing the exact type or subtype of notification of each service to subscribe to, but this is a limitation set to simplify the graphical user interface. The notification feed is a vertical time-line where new notifications are placed on top, much like in social media. Notifications are filtered on either service alone (by tapping from the unfiltered list, which is seen in Figure 5) or both service and notification type together (by tapping from the service filtered list). The app allows the broker address to be customised.

In terms of connecting to and communicating with the ActiveMQ broker, our app utilises the binary MQ Telemetry Transport (MQTT) protocol [21]. MQTT is designed with a simple API, a fixed header of just 2 bytes length and a light keep-alive mechanism. MQTT is with its very small overhead especially suitable for small-footprint devices, as witnessed by its use in Facebook's mobile messenger application [22]. MQTT has support for publish-subscribe, and its community develops client libraries for several platforms. When our notification app was implemented, the most popular MQTT client library was the Java-based *MQTT-Client* library from Fusesource [23]. This library does however not support TLS, but since integration worked out-of-the box, we decided to use it for our initial implementation. The MQTT connection is handled by a service, which maintains the connection even if the app is not running in the foreground.

The app creates subscription strings for topics in the broker based on service ID, and receives notifications as soon as they are published. The performance can however depend on the available data connection speed, as the case always is for mobile devices. If the mobile client has been offline, any notifications that have been published during this time-frame will be received immediately upon re-connection. This is achieved with durable subscriptions (using a customisable time-out). The notifications are delivered according to the quality of service specified when either establishing the connection or publishing the message, depending on the protocol used. In order to identify anonymous subscribers between sessions, the app identifies itself with the unique device ID provided by the operating system, each time it connects to the broker.

IV. Discussion

The challenges of the notification infrastructure presented in this paper are very much similar to the ones ENISA [24] has identified for the European Union (EU) with data breach notification requirements for the electronic communications sector, in the ePrivacy Directive [3]. The bullets below explain our view on how these are or should be handled:

 ISBN: 978-1-61208-378-0

- *Risk prioritisation: The seriousness of a breach should determine the level of response, and breaches should be categorised according to specific risk levels.* With our solution, the first prioritisation is done by the receiver when he determines which notifications he wants to subscribe to—as is the nature of a publish-subscribe architecture. Secondly, there are dedicated notifications for changes in threat level, and the receiver can specify threshold values for when corrective actions should be applied. If the receiver has missed out on subscribing to certain threats these notifications will never be received. This is a weakness in cases where entirely new threats appear.
- *Communication channels: Operators want assurances that notification requirements will not negatively impact their brands.* Here, the objectives should be to prevent tampering of messages and ensure that false breach reports are not submitted. The current implementation of our notification infrastructure have several weaknesses, e.g., there is only a weak authentication scheme for the notification source, and notifications are not signed. However, the authentication scheme could be improved by configuring LDAP or JAAS on the ActiveMQ broker, and the encryption of the messages exchanged with the mobile client could be achieved by using a MQTT client library with TLS support, e.g., IBM's MQTT SDK [25]. A comprehensive list of requirements for a secure logging infrastructure has further been described by the Common Event Expression (CEE) project [26], which should be taken into account in future work.
- *Resources: Budgetary allocations for regulatory authorities should reflect new regulatory responsibilities.* The notification infrastructure itself is cloud deployable and can be scaled up and down according to needs. The cost of brokering notifications is very low. If the reporting from service providers is handled automatically by sensors, this cost is negligible as well, while involving manual human labour increases cost. These costs could in turn be reduced with participation from Computer Security Incident Response Teams (CIRTs), collaborating across sectors or even across borders, such as within the EU.
- *Enforcement: Data controllers will be less incentivised to comply with regulations if regulatory authorities do not have sufficient sanctioning powers.* In addition to regulatory authorities, sending out breach notifications should be motivated by contractual terms between the service provider and consumer, audits from an independent third party, as well as a genuine desire to achieve trust by being open. A late disclosure of incidents will in many cases damage the reputation of a service provider more than the incident itself.
- *Undue delay in reporting: Regulatory authorities want to see a short deadline for reporting breaches to authorities and data subjects [..] Service providers, however, want their resources to be focused on identifying if the problem is serious and solving the problem, instead of spending time reporting details, often prematurely, to regulatory authorities*. Our notifications are primarily short messages that can be distributed rapidly. Due to the nature of publish-subscribe, messages will be delivered as soon as they are published and the subscriber is online. Such messages takes little time and effort to create, especially if done automatically. Therefore, delays are not considered a major obstacle. Within the EU, telecommunications operators and ISP providers must inform national authorities within 24 hours after breach detection with at least an initial set of information, with more details to follow within three days [27].
- *Content of notifications: Operators want to make sure that the content of the notifications does not impact negatively on customer relations. Regulatory authorities, however, want to see that the notifications provide the necessary information and guidance in line with the rights of the data subjects.* As stated above, our messages as short, early warnings that do not contain much information by themselves. More detailed content can be sent out at later stage through other means. Since we can send notifications wrapped entirely as JSON objects, we are also able to extend the value of the notification messages as a JSON object with additional properties to enable customisation as needed. A potential threat to all notification systems is related to fake notification and manipulation of reputation based systems. We refer to one of our previously published paper for a deeper discussion on this [28].

A few related efforts have been made on standardising security event message content and formats, but we are unaware of efforts with the purpose of supporting automated service composition. While our work started in the experimental end with self-adaptable service compositions in mind, standardisation is needed at some point. There will definitively be potential to learn from similar initiatives, however at the time of writing there seems to be little activity in the area. The already mentioned CEE project [26] was for instance initiated to standardise event descriptions to support auditing and users' ability to comprehend event log and audit data. CEE defines both delivery methods and filtering, as well as an event structure—although in a flat manner. CEE is extensible in a way that can redefine any part of the taxonomy, although that might not be a good thing for a publish-subscribe infrastructure. The project has however been shut down due to a lack of financial support. The Distributed Audit Services (XDAS) specification [29] was similarly created to support the principle of accountability and detection of security policy violations in distributed systems. XDAS defines a taxonomy of events categories (layers) comprising varying levels of semantics and context, but is very focussed on recording events for correlating audit trails. The specification is hence largely targeted at auditing and compliance, and has not become a formally approved standard. Work on XDAS v2 appears to have been initiated in recent years, but without publishing any significant progress.

Having a notification infrastructure by itself is obviously of little value if there are neither agents creating notifications nor anyone receiving them. Together with academia and industry (18 partners in total), we have evaluated the API,

integration and performance with others tools for monitoring and adaptation service-oriented systems. Results from this evaluation shows high levels of satisfaction for installation, documentation, integration and stability [30]. Our focus for future work evolves around integration with new tools and securing the infrastructure itself.

V. Conclusion

We have demonstrated a distributed notification infrastructure that facilitates runtime management and adaptation of service compositions. Though a service component may be regarded as reliable and secure enough when the composite service is designed, its security and privacy properties and attributes for quality of service can change during its life-time. In addition, risks are not static, and threats and vulnerabilities in service components can impact the overall security level of a composite service. Due to regulatory pressure and a need for trustworthiness through accountability in service composition chains, we believe that the concept of a common notification infrastructure is needed. Further work is however needed in terms of securing the infrastructure, research is needed on how it could be managed in practice, and standardisation work is needed to agree on notification content descriptions.

Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grants no 257930 (Aniketos), 317631 (OPTET) and 371550 (A4Cloud). The authors are not affiliated with any of the service providers referenced in the use case, and the events described are purely hypothetical.

References

[1] S. Pearson et al., "Accountability for cloud and other future Internet services," 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Dec 2012, pp. 629–632.

[2] E. A. Gjære and P. H. Meland, "Threats management throughout the software service life-cycle," EPTCS, vol. 148, 2014, p. 114.

[3] European Commission, "Directive on privacy and electronic communications," Tech. Rep., 2002.

[4] National Conference of State Legislatures. Security breach notification laws. [Online]. Available: http://www.ncsl.org/research/telecommunications-and-information-technology/security-breach-notification-laws.aspx [retrieved: September, 2014]

[5] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 35–44. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806809

[6] Aniketos project. AniketosEU on GitHub. [Online]. Available: https://github.com/AniketosEU [retrieved: September, 2014]

[7] ECMA International, ECMA-404 The JSON Data Interchange Standard, Std., October 2013.

[8] Object Management Group, Business Process Model and Notation (BPMN) Version 2.0, Std., January 2011.

[9] P. H. Meland and E. A. Gjære, "Threat Representation Methods for Composite Service Process Models," International Journal of Secure Software Engineering, vol. 4, no. 2, 2013, pp. 1–18.

[10] M. Salnitri, E. Paja, and P. Giorgini, "Preserving compliance with security requirements in socio-technical systems," in Proceeding of Cyber Security and Privacy (CSP) forum 2014, 2014.

[11] FinansCERT. Norwegian financial sector cybercrime unit. [Online]. Available: http://www.finanscert.no/engelsk.html [retrieved: September, 2014]

[12] Riku, Antti, Matti and Neel Mehta. Heartbleed bug. [Online]. Available: http://heartbleed.com/ [retrieved: September, 2014]

[13] MITRE Corporation. Common vulnerabilities and exposures (cve) database. [Online]. Available: http://cve.mitre.com/ [retrieved: September, 2014]

[14] Aniketos project, "Deliverable D4.3 Algorithms for responding to changes and threats," Tech. Rep., August 2013.

[15] B. Zhou et al., "Secure service composition adaptation based on simulated annealing," in 6th Layered Assurance Workshop, 2012, p. 49.

[16] S. Chatterjee. Messaging Patterns in Service-Oriented Architecture, Part 1. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa480027.aspx [retrieved: September, 2014]

[17] The Apache Software Foundation. Apache ActiveMQ website. [Online]. Available: http://activemq.apache.org/ [retrieved: December, 2014]

[18] Oracle, Java Message Service Specification, Std., November 1999.

[19] Oracle. Java Authentication and Authorization Service (JAAS) Reference Guide. [Online]. Available: http://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html [retrieved: September, 2014]

[20] Activiti. Activiti bpm platform website. [Online]. Available: http://www.activiti.org/ [retrieved: September, 2014]

[21] IBM and Eurotech, MQTT V3.1 Protocol Specification, Std., July 2014. [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[22] L. Zhang. Building Facebook Messenger. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920 [retrieved: September, 2014]

[23] Fusesource. MQTT-Client An Open Source Java MQTT v3.1 Client. [Online]. Available: http://mqtt-client.fusesource.org/index.html [retrieved: September, 2014]

[24] S. Górniak et al., "Data breach notifications in the eu," ENISA, Tech. Rep., 2011.

[25] IBM Corporation. Getting started with the MQTT client for Java on Android. [Online]. Available: http://www-01.ibm.com/support/knowledgecenter/SS9D84_1.0.0/com.ibm.mm.tc.doc/tc10130_.htm [retrieved: September, 2014]

[26] MITRE Corporation, "CEE Log Transport (CLT) Specification," Tech. Rep., 2012. [Online]. Available: https://cee.mitre.org/language/1.0-beta1/clt.html

[27] European Commission. Digital Agenda: New specific rules for consumers when telecoms personal data is lost or stolen in EU. [Online]. Available: http://europa.eu/rapid/press-release_IP-13-591_en.htm [retrieved: September, 2014]

[28] P. H. Meland, "Service injection: A threat to self-managed complex systems," in Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, Dec 2011, pp. 1–6.

[29] The Open Group, Distributed Audit Services (XDAS), Std., January 1997.

[30] Aniketos project, "Deliverable D7.3 Results of the final validation and evaluation of the ANIKETOS platform," Tech. Rep., May 2014.

 ISBN: 978-1-61208-378-0

A Policy-based Middleware for Self-Adaptive Distributed Systems

Jingtao Sun and Ichiro Satoh

Department of Informatics. School of Multidisciplinary Science
The Graduate University for Advanced Studies
National Institute of Informatics
Tokyo, Japan
Email: {sun, ichiro}@nii.ac.jp

***Abstract*—This paper presents a novel approach to dynamically adapting distributed applications to changes in environmental conditions. e.g., available network resources and users requirements. The key idea behind the approach is to introduce the relocation of software components to define functions between computers, as a basic mechanism for dynamic adaptation on distributed systems. It also introduces application-specific built-in policies for relocating software components to define high-level adaptation by human-readable declarative policy scripts. It is constructed as a middleware system for Java-based general-purposed software components. This paper describes the design and implementation of the approach with several applications, e.g., remote information retrieval and distributed media service.**

Keywords–software component; policy-based; self-adaptive; middleware; Distributed System; architecture-level.

I. INTRODUCTION

Distributed systems are essentially dynamic in the sense that computers and applications may be dynamically added to or removed from them or networks between computers may be disconnected or reconnected, dynamically. Therefore, the running of software components of which an application consists, should be depended by nature, so that the systems can adapt to various changes at component runtime systems. On the other hand, the running of software components on a distributed system should be adapted to and reuse them on different distributed systems.

Distributed applications are executed for multiple-purposes and multiple users whose requirements may change in various cases. However, on a variety of distributed systems, their structure may also changes. Adaptation must support the variety and change in the underlying systems and the requirements of applications should be separated from business logic. Therefore, we distinguish adaptation concerns and business logic concerns by using the principle of separation of concerns so that developers for applications can concentrate their business logic rather than adaptation as mush as possible. A solution to this is to introduce concern-specific languages for separating adaptive concerns from business logic concerns. There have been several attempts [3][7] to support the separation of concerns on non-distributed systems, but adaptation mechanisms in distributed systems tend to be complicated, so that it is difficult to define primitive adaptation.

This paper addresses the separation of adaptation concerns from application-specific logic concerns in distributed systems. We assumed that a distributed application would consist of one or more software components, which might run on different computers through networks. Our proposed approach has two key points. The first is to introduce policies for relocating software components as a basic adaptation mechanism. The second is to provide nature-inspired relocation policies for application-specific adaptations. When the changes have occurred, e.g., in the requirements of applications and the structures of system, its software components would automatically be relocated to different computers according to their policies to adapt to changes. We are constructing a middleware system for building and operating self-adaptive distributed systems.

The proposed approach is based on adaptive deployment of software components, but is different from other existing works [2][6][7], the functions of which are inside software component. If this components are adapted, other component may have serious problems. For example, they can not communication with the adapted ones. On the other hand, the relocation of software components do not lose potential functions of components. This problem may seem to be simple, but it makes their applications resilient without losing availability, dependability, and reliability. In fact, our approach can provide adaptation between general-proposed approaches in distributed systems.

This paper focuses on the middleware we have developed to simplify the design and deployment of policy-based runtime system in real applications. Section 2 describes the requirements of distributed systems and gives readers the key idea behind the proposed approach. Section 3 gives an overview of how to design the policy-based middleware and what kind of adaptations are driven by declarative policy scripts. Following this, Section 4 describes how we implemented such runtime system support software components. Section 5 describes several applications of this middleware to demonstrate its strengths. Section 6 describes related work, with conclusions and planned future work described in Section 7.

II. APPROACH

In distributed systems, the requirements of applications or users and the environments of distributed systems often change, so they have to adapt to these changes inside of themselves. But in real environments, applications should not be necessary to recompile one more again in order to adapt to use a different network architecture or layout. This is because most of existing adaptation technology needs a large amount of resources to adapt the requirements of the applications or adaptation is limited or adaptive contents cannot be predicted. Therefore, we decided to propose a novel approach to relocate the running software components form one computer to another one, to adapt to changes for distributed systems,

 ISBN: 978-1-61208-378-0

e.g., adapting to distributed systems, networks architectures or available resources.

A. Requirements

Most existing distributed systems have been statically constructed based on several types of system architectures, e.g., client-server, peer-to-peer and master-slave according to their original requirements. However, with the development of in-depth in distributed systems, some of the requirements may be changed, wherefore the distributed systems need to dynamically adapting themselves. For example, computers and networks may have failures or some new computers may be added to or removed from the networks, or the requirements of applications, which may be running on different computers. Therefore, distributed systems need some abilities to adapt to such changes. Furthermore, to support drastic changes in system structures, distributed system architectures themselves require to change.

In this section, we describe the requirements of our policy-based middleware as follows:

- Self-adaptation: Distributed systems essentially lack no global view due to communication latency between computers. Software components, which may be running on different computers, need to coordinate them to support their applications with partial knowledge.
- Separation of concerns: software components of which an application consist should be defined independently in our adaptation mechanism. On the other hand, these software components will deploy themselves to destination computers, according to the predefined policies or user-defined policies, which can be developed by system operators or be automatically executed by themselves.
- General-purpose: Various of applications are running on distributed systems. Therefore, our adaptation mechanism should be implemented as a practical middleware to support general-purpose applications.
- Reduce Input/Output cost: The costs of Input/Output handling are huge in distributed systems, e.g., when users send reading requests to front-ends servers, they have to find out the requested files in the first, and then reading the content of the files line by line, until the ends of files. For this reason, the cost of Input/Output requires to be reduced.
- Dependability: In order to improve the dependability of distributed systems, middleware systems do not require to support a centralized management for software components and make sure that data keep their consistency.

Computers on distributed systems may have limited resources, e.g., processing, storage resources, and network architectures. Therefore, our approach should be available with such limited resources, whereas many existing adaptation approaches explicitly or implicitly assume that their targets of distributed systems have enriched resources.

B. Adaptation

Our approach separates software components from their policies for adaptation. This is because the user-defined policies can be reused to reduce the cost of compiling themselves once more.

1) Deployable software components: Generally, an application consists of one or more software components, which may be running on different computers. Therefore, in our approach, these components can be deployed at other computers, according to its deployment policies. It is defined as a collection of Java objects in the current implementation. It also has an interface, which called references. By executing them, soft components can migrate to destination computers, and then communicate with the destination components through dynamic methods invocation.

2) Deployment policy for adaptation: Each component can have one or more policies, where each policy is basically defined as a pair of information on where and when the component is deployed. Before explaining deployment policies in the proposed approach, we have to discuss policy scripts for adaptation on distributed systems. We describe these concepts as follows:

- This approach does not support any adaptation inside software components. Because software components should be general-purposed and adaptation-independent.
- Each component has one or more policies, where policy specifies the relocation of their components and instructs them to migrate to destination computers, according to specified conditions.
- Each policy specifies as a pair of a condition part and at most one destination part. The former is written in a first-order predicate logic-like notation, where predicates reflect information about the system and application. The latter refers to another component instead of itself. This is because such policies should be abstracted away from the underlying systems.

C. Destination of policies

Under the user-defined destination of policies, as Figure 1 shows, software components can be dynamically deployed at destination computers and the destinations of policies can be easily changed for reuse by other distributed systems. The policies are described as follows:

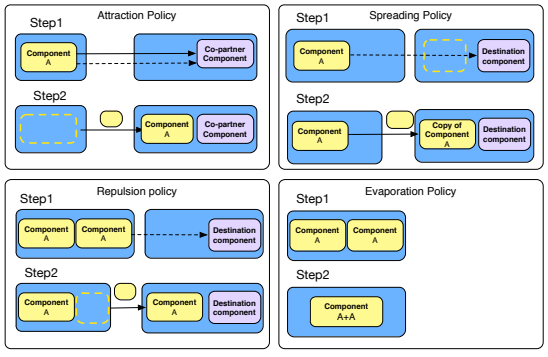

Figure 1. Destination of policies.

- Attraction: Frequent communication between two components yields stronger force. The both or one of the components are dynamically deployed at computers that other computers are located at.

 ISBN: 978-1-61208-378-0

- Spreading: Copies of software components are dynamically deployed at neighboring and propagated from one computer to another over a distributed system. This policy progressively spreads components for defining functions over the system and dynamically adds to the lack of the functions.
- Repulsion: Computers are deployed at computers in a decentralized manner to avoid collisions among them. This policy migrates software components from regions with high concentrations of components to low concentrations.
- Evaporation: Excess of components results in overloads. The same or compatible functions must be distributively processed to reduce the amount of load and information. The policy consists in locally applying to synthesize multiple components or periodically to reduce the relevance of functions.

This approach assumes that an application consists of one or more software components, which provide their own functions and may be running on different computers. It introduces the adaptive deployment of software components but not of adaptive functions inside software components. If functions inside software components are adapted, other components, which communicate with the adapted ones, may have serious problems. However, the relocation of software components does not lose the potential functions of components. This may seem to be simple but it makes their applications resilient without losing availability, dependability, or reliability. It can also separate adaptation from components, which define application logic, because components themselves are defined and executed independently of any adaptation.

III. DESIGN

The proposed approach dynamically deploys components to define application-specific functions at computers according to the policies of the components to adapt distributed applications to changes on distributed systems.

A. Dynamically adaptive system architecture

Our middleware architecture consists of three parts (Figure 2). The first part is the adaptation manager, the second part is runtime system and the third part is distributed applications. From this architecture, we can notice that:

- The first part is a component runtime system, which is responsible for executing software components, migrating software components and enabling them to invoke methods at other software components. However, by using these methods, the software components need to be serialized in the first, then migrate themselves from one computer to others. When these software components arrived at destination computers, they can communicate with the components of destination computers, according to naming inspection.
- The second part is adaptation manager, which is responsible for our runtime system. However, it can control the behaviour of components, select policies from destination database and fetch them and determine themselves where the software components should be moved. The policies are written by an Event-Condition-Action format scripting language.

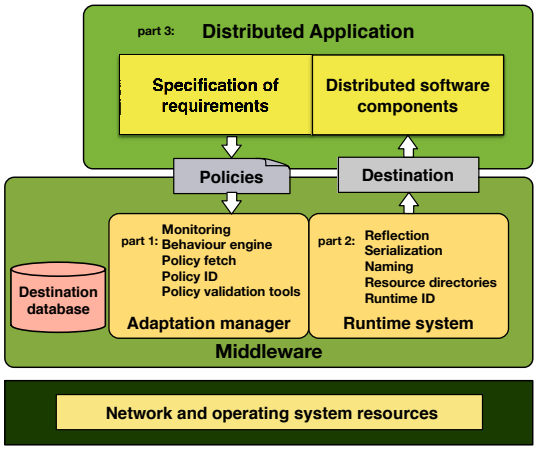

Figure 2. Middleware architecture.

- The third part consists in distributed applications, which can be designed by any general-purposes.

B. Component runtime system

Every runtime system allows each component to have at most one activity through the Java thread library. When the life-cycle state of a component is changed, e.g., when it create, terminates, duplicate, or migrates to another computer. The runtime system issues spcific events to the software component. To capture such events, each component can have more than one listener object that implements a specific listener interface to hook certain events issued before or after changes have been made in its life-cycle state. Through this method, we can easily hide the differences between the interfaces of objects at the original and other computers. Each runtime system can exchange components with other runtime system through TCP channels by using Object Input/Output Stream. When a component is transferred over networks, not only the code of the components, but also their state can be transmitted into a bit stream by using Java's object serialization package, and then the bit stream is transferred to the destination computers. The runtime system which is run on the receiving side receives and unmarshals the bit stream. When components have been deployed at destination computers, their methods should still be able to be invoked from other computers, which are running at local or remote computers.

C. Adaptation mechanism

The policy-based deployment of components is managed by adaptation managers, where they are running with software component runtime systems and they have not any centralized management servers. Each component runtime system periodically advertises its address to other runtime systems through UDP multicasting, and then these computers can return their addresses and capabilities to the destination computers through TCP channels. Each policy is specified as a pair of conditions and actions. The former is written in a first-order predicate logic-like notation and its predicates reflect various system and network properties, e.g., network connections and application-specific conditions and the utility rates and processing capabilities of processors. The latter is specified as a relocation of

 ISBN: 978-1-61208-378-0

components. Our adaptation was intended to be specified in a rule-style notation.

1) Adaptation policy specification format: The adaptation manager offers an interpreter to execute the specification format. Since we need to predict conflicts and divergences that result from adaptations, we need to design a format for specifying adaptations policies, which are given as the relocation of components according to changes in their systems and the requirements of their applications. The format consists of conditions at destination parts. The two parts are defined based on a theoretical foundation to verify the validation of adaptations. The former is written in a first-order predicate logic-like notation, where predicates reflect information about the system and applications. e.g., the utility rates and processing capabilities of processors, network connections, and application-specific conditions. The latter represents the deployment and duplication of components in our adaptation instead of any application-specific behaviors, including communications and state transition, of the components. It is formulated as a process calculus.

Since policies are written in a XML format (as Figure 3 shows), it can be defined outside components. In addition, these user-defined policies can be reused for other components, and the components can be reused with other policies. For now, our adaptation manager provides four built-in policies. Each policy contains [Event-Condition-Action] three main tags. We can define the name of this policy in *Event* tag. The *Condition* tag shows when the software components should be migrated or not. It can be freely defined by the users or developers of distributed systems. Our interpreter is different from the existing researches [13][15], because our approach is focused on components migration, so in *Action* tag, we can judge the software components where to relocate or whether to migrate or not.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy id = >
   <!-- Event of Policy  -->
        <Event>
              <Name>  arir1  </Name>
        </Event>
        <!-- Condition of Policy.e.g., predicate1,...predicaten  -->
        <Condition>
           adaptive_file_size >= max_file_size ll
                 adaptive_bandwidth >= fixed_value
        </Condition>
        <!-- The destination of relocation  -->
        <Action>
              <destination>
              If (condition == true){
                    migrate (component 1,...,component N,
                         destination_IP, destination_port )
                    recall (component 1,...,component N,
                         method_name )
                 } else {
                    unMigrate ()
                     }
              </destination>
        </Action>
</</Policy>
```

Figure 3. Policy format.

For example, to adapt remote information retrieval. In this sense, we can define an attraction policy and execute this format in adaptation manager as follows:

- When a component has an attraction policy for another component, if the condition specified in the policy is satisfied, the policy instructs the the former to migrate to the current computer of the latter.
- When a component has a spreading policy, the policy will make a copy of the component and instructs the copy to migrate to the current computer.
- When a component has a repulsion policy for another component, if this computer have the same or compatible components, this policy will migrate this component which communication with another component to the current computer.
- When a component has an evaporation policy, if the condition specified in the policy is satisfied, it terminates.

When the external system detects changes in environmental conditions, the runtime system can self-adaptive to migrate the search component to remote computer. If this remote computer is failure or waiting processing in threads, the search components can relocate to other computers, according to the user-defined policies. Then, the search component can fetch files inside of itself. Once the retrieval completed, the search component will return back, according to the attraction policy. However, the details will be described in Section 5.

IV. IMPLEMENTATION

This section describes the current implementation of a middleware system based on the proposed approach.

A. Component runtime system

Each component is a general-purpose and programmable entity, which defined as a collection of Java objects and packaged in the standard JAR file format. It can migrate and duplicate themselves between computers. Our runtime system is similar to a mobile agent platform, but it has been constructed independently of any existing middleware systems. This is because existing middleware systems, including mobile agents and distributed objects, have not supported the policy-based relocation of software components. Our middleware system is built on the Java Virtual Machine (JVM), so it can abstract away between different operating systems.

The current implementation basically uses the Java object serialization package to marshal or duplicate components. The package dose not support the capture of stack frames of threads. Instead, when a component is duplicated, the runtime system issues events to it to invoke their specified methods, which should be executed before the component is duplicated or migrated. Furthermore, this system suspends their active threads. We also implement this system by using our original remote method invocation between computers instead of Java Remote Method Invocation (RMI); this is because Java RMI dose not support object migration.

B. Adaptation manager

The adaptation manager is running on each computer and consists of two parts: a database of policies and an event manager. The former will compile and execute user-defined policies and the latter will receive events from the external systems and notify changes in the underlying systems and applications.

 ISBN: 978-1-61208-378-0

We describe a process of the relocation of a software component, according to user-defined policies.

- When a component creates and arrives at a computer, it automatically registers its deployment policies with the database of the current adaptation manager.
- The manager periodically evaluates the conditions of the policies maintained in its database.
- When it detects the policies whose conditions are satisfied, it deploys software components at destination computer, according to the selected policies migrate component to the destination computer and dynamically invoke the methods of destination software components.

Two or more policies may specify on different destination computers, under the same condition that drive them. The current implementation provides no mechanism to solve conflict between policies. So, we assume that policies would be defined without any conflicts right now. The destination components may enter divergence or vibration models due to conflicts between component policies. In addition, they may have multiple deployment policies. However, the current implementation dose not exclude such divergence or vibration.

V. APPLICATIONS

This section describes two applications of the proposed approach.

A. Adaptive remote information retrieval

Suppose users who are using search engines to find certain text patterns from data located at remote computers like Unix's grep command. A typical approach is to fetch files from remote computers and locally find the patterns from all the lines of the files. However, if the sum of the volume of its result or the size of a component for searching patterns from data is bigger than the volume of target data, the approach is efficient. In this case, the components should be executed at remote computers that maintain the target data rather than at local computers. However, it is difficult to select where the components is to be executed because the volume of the result may be not known before. The proposed approach can solve this problem by relocating such components from local computers to remote computers while they are running. Figure 4 shows our system for adaptive remote information retrieval, which consists of client, search, and data access manager components. The client component and the data access manager component are stationary components. The search component supports finding text lines that match certain patterns provided from the client component in text files that it accesses via the data access manager component. It has an attraction policy that relocates itself from local to remote computers when the volume of its middle result is larger than the size of component; otherwise, it relocates from remote to local computers.

Although the volume of the result depends on the content of the target files and the patterns, it is typically about one over hundred less than the volume of the target files. Therefore, the cost of our system is more efficient in comparison with Unix's grep command. This means that our approach enables distributed application to be available with limited resources and networks. Our approach is self-adaptive in the sense that it enables the search component to have its own adaptation

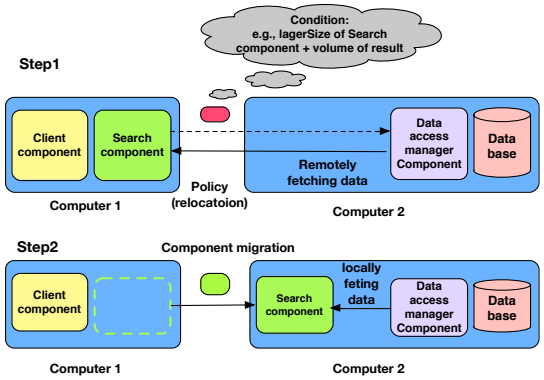

Figure 4. Adaptive remote information retrieval.

policy and manages itself according to the policy independently of these components themselves. It is independent of its underlying systems because the destination of our component relocation is specified as components, instead of computers themselves and they can be reused also. Furthermore, when the remote or local computer is failure, our system can migrate the retrieval programs to another computer, according to the user-defined policies for go on progressing without termination processes. This is the difference between traditional approaches and ours.

B. Adaptive distributed media service

Suppose a media distribution service, including video streaming [17]. Because of the sizes of media contents tend to be large recently, so that the cost of transmitting such contents from back-end servers to front-end servers and from front-end servers to clients becomes huge. Therefore, it is necessary in the vicinity of the data processing to save the high costs of data transmission.

We assume that users send requests to front-end servers, and convert video data via front-end servers before fetching them. Figure 5 shows our system for adapting distributed media service, which consists of client, search, Drawing UI component, and data access manager components. The client component and the data access manager component are stationary components. The search component supports to response the requests from users. When the tasks becomes excessive in a short time, the front-end server will become a bottleneck. The Drawing UI component supports to adapt to the size of the screens.

In this case, clients may fail to connect to or have to wait while data have processed by front-end servers. Therefore, the front-end servers components should be dynamically deployed at the back-end servers and processed at the back-end ones on behalf of the front-end ones. If the back-end servers have enough resources for processing. The contents at back-end servers do not need to be relocated and copied. This approach can reduce the cost of transmitting the content from the back-end servers to the front-end servers so that it is useful to avoid heavy traffic between the servers.

On the other hand, we assume that users try to select media contents from front-end servers, because they are responsible for managing the selections of contents and drawing UIs for

 ISBN: 978-1-61208-378-0

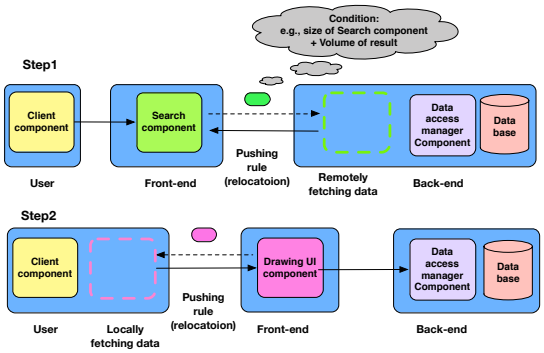

Figure 5. Adaptive distributed media service.

users to do. By using our approach, we can relocate the running components for selecting media from front-end servers to clients when clients have much capability to manage and draw UIs.

These deployments of software components can be specified in our policy-based middleware for adaptations and automatically invoked methods of destination components when conditions of policies are satisfied.

VI. Experiment

In this paper, we present the implementation of the proposed system on OS X, which has Intel Core i7 2GHz as CPU and 8GB memory and the download/upload speed of internet is 8.586KB/s and 15.83KB/s. The implementation uses raw socket to obtain all packages and is described in Java programming language. For experiment, this paper prepares a network environment where to adapt remote information retrieval. The experiment in this paper compares the cases use and non-use the proposed adaptation middleware system. In the experiment, we assume that one user retrieves a keyword named JAVA and we searched the keyword in three types of files.

A. Result of Experiment

The first one type of file has the data size of 17KB. The second one that the size of data is 1.1MB. The data size of the third one is 104MB. In addition, we compare the speed of search the same keyword in our system. Figure 6 shows the result of experiment for adapting remote information retrieval.

Keyword	File size	With adaptation	Without adaptation
JAVA	17KB	2.11sec	1.39sec
JAVA	1.1MB	4.62sec	4.97sec
JAVA	104MB	6.17sec	13.12sec

Figure 6. Results of experiment.

By the three sets of data, we know that if the file becomes larger, our self-adaptive middleware system can significantly reduce the time of remote information retrieval. From the first set of data, we know that when the size of retrieved file is almost as large as the retrieval programs, the processing time of our self-adaptive middleware system is longer than non-adaptation system. This is because, the deploy objects of software components in our middleware require serialization/de-serialization between local and remote sides. However, the non-adaptation system does not require its. From the second set of data, we know that when the size of retrieved file is almost 10 times larger than the retrieval programs, our system is slightly stronger than without adapted remote information retrieval. However, by the last set of data, we clearly know that when the size of the file becomes 100 times, the search time of our middleware system will spend half time of non-adaptation information retrieval. Furthermore, we can not only reduce remote information retrieval time through our approach, we can also enhance distributed systems reliability, dependability ,and availability. For example, when the remote computer fails, our system will temporarily freeze the retrieval programs, and migrate its to another computer. Then these programs will be thawed and resumed in remote side.

VII. Related Work

This section describes a selection of related research in the fields of distributed systems. It compares our approach with several existing adaptation approaches for distributed systems.

Many researchers have explored adaptation mechanisms for distributed systems [4][5][18]. They can be classified into three types. The first is to dynamically change coordination between programs, which are running on different computers for their adaptation, e.g., CORBA-based middleware [6][7][8][10]. This is one of the most typical adaptive coordination that enables client-side objects to automatically select and invoke server-side objects according to changes in their requirements of applications or system architectures. However, this type is limited. Because it only modifies the relationships between distributed programs instead of the computers, which are executing them. The second is to change programs for defining functions of which an applications consists, e.g., genetic programming [11]. It needs more resources to select generations of programs. On the other hand, it is difficult to predict their adaptation. Distributed systems should be predictable because they are often used for mission-critical applications. The third type is policy-based middleware on distributed systems [2][3][12][16]. By using policies to define the conditions of software components, these approaches can migrate the components to specified computers by a specified adaptation language, seems like [1][13][15]. However, the specified computers may be not a good choose. Because the specified computers may be have not enough available resources or the processing of threads are waiting several task or the connection has been broken. Conversely, our proposed approach can autonomously select the destination of deployed software components. Therefore, they do not care the computer is a specified one or not. It is a more general-purpose.

On the other hand, our approach can change the computers that execute programs for self-adaptation. Therefore, it enables the programs to escape from computers, which may have system failures or be shutdown.

The relocation of software components have been studied in the literature on mobile agents [9][14]. By using the technology, we may be able to dynamically relocate the executing programs of components. Furthermore, like ours, several

 ISBN: 978-1-61208-378-0

mobile agent platforms support mechanisms for mobility-transparence, where the mechanisms enables programs, which may be migrated to remote computers to continue to work on other computers. However, the technology itself does not intend to support adaptation so that it cannot abstract away adaptation from application-developers. Furthermore, our approach is inherently designed for dynamic adaptation on distributed systems.

VIII. CONCLUSION

This paper proposed an approach to adapt distributed applications by predefined or user-defined policies. It introduced the relocation of software components between computers as a basic mechanism for adaptation. It separated software components from their adaptations in addition to underlying systems by specifying policies outside the components. It is simple, but it provided various adaptations to support resilient distributed systems without any centralized management. It was available with limited resources because it had no speculative approaches, which tended to spend computational resources. The relocation of components may have security problems, e.g., executions malicious programs, but it is can be solved by receiving only the programs that are transmitted from secure and reliable computers with authentication techniques. It was constructed as a general-purpose middleware system on distributed systems instead of any simulation- based systems. Components could be composed from Java objects like JavaBean modules. We described several approaches with practical applications.

REFERENCES

[1] J.-X. Li, B. Li, L. Li and T.-S. Che, "A policy language for adaptive web services security framework." In: Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. Eighth ACIS International Conference on. IEEE, 2007. pp. 261-266.

[2] J. Keeney and V. Cahill, "Chisel: A policy-driven, context-aware, dynamic adaptation framework." Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on. IEEE, 2003. pp. 3-14.

[3] K. Yang, G. Alex and T. Chris, "Policy-based active grid management architecture." Networks, 2002. ICON 2002. 10th IEEE International Conference on. IEEE, 2002. pp. 243-248.

[4] A. Tripathi, "Challenges designing next-generation middleware systems." Communications of the ACM 45.6 ,2002, pp. 39-42.

[5] V. Issarny, C. Mauro and G. Nikolaos, "A perspective on the future of middleware-based software engineering." 2007 Future of Software Engineering. IEEE Computer Society, 2007. pp. 244-258.

[6] B. Gordon S, B. Lynne, I. Valerie, T. Petr and Z. Apostolos, "The role of software architecture in constraining adaptation in component-based middleware platforms." Middleware 2000. Springer Berlin Heidelberg, 2000. pp. 164-184.

[7] D. Kulkarni and T. Anand, "A framework for programming robust context-aware applications." Software Engineering, IEEE Transactions on 36.2. 2010. pp. 184-197.

[8] R. Olejnik, B. Amer and T. Bernard, "An object observation for a Java adaptative distributed application platform." Parallel Computing in Electrical Engineering, 2002. PARELEC'02. Proceedings. International Conference on. IEEE, 2002. pp. 171-176.

[9] I. Satoh, "Mobile agents." Handbook of Ambient Intelligence and Smart Environments. Springer US, 2010. pp.771-791.

[10] J. Zhang and B. H. Cheng, "Model-based development of dynamically adaptive software." Proceedings of the 28th international conference on Software engineering. ACM, 2006. pp. 371-380.

[11] Koza and R. John, "Genetic programming: on the programming of computers by means of natural selection." Vol. 1. MIT press, 1992.

[12] M. Luckey and E. Gregor, "High-quality specification of self-adaptive software systems." Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, 2013. pp. 143-152.

[13] Anthony and J. Richard, "A policy-definition language and prototype implementation library for policy-based autonomic systems." Autonomic Computing, 2006. ICAC'06. IEEE International Conference on. IEEE, 2006. pp. 265-276.

[14] R. Montanari, T. Gianluca and S. Cesare, "A policy-based mobile agent infrastructure." Applications and the Internet, 2003. Proceedings. 2003 Symposium on. IEEE, 2003. pp. 370-379.

[15] N. Damianou, D. Naranker, L. Emil and S. Morris, "The ponder policy specification language." Policies for Distributed Systems and Networks. Springer Berlin Heidelberg, 2001. pp. 18-38.

[16] D. Ferraiolo and G. Serban, "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement." Journal of Systems Architecture 57.4. 2011. pp. 412-424.

[17] Z.-J. Lei and D. G. Nicolas, "Context-based media adaptation in pervasive computing." Electrical and Computer Engineering, 2001. Canadian Conference on. Vol. 2. IEEE, 2001. pp. 913-918.

[18] A. Uribarren, J. Parra1, R. Iglesias1, J. P. Uribe1 and D. Lopez-de-Ipina, "A middleware platform for application configuration, adaptation and interoperability." Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on. IEEE, 2008. pp. 162-167.

 ISBN: 978-1-61208-378-0

SLA Object and SLA Process Modelling using WSLA and BPM Notations

Towards defining a Generic SLA Orchestrator Framework

Bukhary Ikhwan Ismail, Nurliyana Muty, Mohammad Fairus Khalid, Ong Hong Hoe
Advanced Computing Lab
MIMOS Berhad
Kuala Lumpur, Malaysia
ikhwan.ismail@mimos.my, nurliyana.muty@mimos.my, fairus.khalid@mimos.my, hh.ong@mimos.my

***Abstract*— SLA monitoring and enforcement ensure service dependability. In an ever challenging market, service providers strive to compete in the IT business by offering new innovative services. Fast to market and reliable Quality of Service goes hand in hand in determining the market leader. To address these challenges, a flexible and comprehensive tool is required to automate the provisioning of SLA and Service Level Management processes. Based on our investigation there is no single framework, which is flexible enough to orchestrate SLA provisioning for multiple services. To address this problem, we first describe the SLA object concepts and redefine the SLA management processes. It introduces an extended WSLA model and tiering mechanism, to promote modularity and reusability of the SLA model. We propose to model the SLA management process using the Business Process Management Notation to simplify and reduce the planning, design, and implementation effort in defining the SLA offerings. Both of these models act as guidelines to attain a generic SLA framework towards any service adaptation.**

Keywords-Service Level Agreement; Service Level Management; WSLA; Business Process Management; Dependable Service

I. INTRODUCTION

Service is a means of delivering value or functions, which the provider offers, to the subscribers. Business applications, software functions and infrastructural services are some examples. Subscribers require a certain level of service delivery guarantee and the Provider needs to ensure the dependability aspect of service fulfilment. In IT Service Management (ITSM), Service Level Agreement (SLA) provides a well-known standard in ensuring service delivery guarantee. The SLA captures the requirements and expectations of service guarantee where both parties agree. The SLA contract formalizes the requirements of; and not limited to Quality of Service (QoS) parameters; responsibilities of both parties; warranties or actions to be taken; compensations; guarantee coverage and service limitations or exclusion clauses [1].

Service Level Management (SLM) is a discipline of proactive methodology and procedures used to ensure adequate levels of service are delivered in accordance with business priorities at an acceptable cost [2]. Most SLA management strategy considers two common phases; (1) the *negotiation of the contract* – the formalisation of objectives, action guarantee and violations and (2) *the monitoring of its fulfilment in real-time* –the proof of service delivery [3].

In this paper, we analyse several prominent research works in the areas of service delivery governance from the provider's perspective. We conclude that 1) *Most SLA frameworks implement on a specific service or specific service domain.* Interpretation of QoS attributes such as availability, reliability, *or* performance is unique to the service domain. For example, storage availability versus web services availability differs greatly. It affects the way the SLA is calculated and action logic to perform. To adapt the same framework for new implementation would be impossible. 2) *The SLA implementation is embedded within service infrastructure* - For example, the SLA rules or logics are buried implicitly in the application code [4]. It would be impossible to utilize the same framework for new implementation. 3) *The dynamics of SLM* - Not all of the process activities e.g., SLA negotiation, template definition and compensation to name a few, are required in order to deliver service level guarantees. Each research project usually defines a fix set of SLM process while in actual implementation, not all of the processes are required. To adapt the same predefined sets of SLA management process for another service is unsuitable.

There are four main contributions of this paper. 1) A survey of multiple SLA research projects, which deduced the importance of SLA concepts, objects and SLM, processes surrounding the SLA management and the governance of service delivery guarantees. 2) A proposed new form of SLA and SLM process meta-model to illustrate the interactions and usage of SLA data objects with the SLM process activities. 3) An extension of existing Web Service Level Agreement (WSLA) model to support additional SLM processes that enable the separation of the *SLA service information*, the *logics* within the SLA, the *implementation* to enforce the SLA and *runtime* information of the model. Lastly, 4) an introduction of a modelling technique using Business Process Management to model the SLM process.

This paper is organized as follows. Section II summarizes the SLM processes. Section III, discusses the requirements and analysis towards the importance of generic SLA framework. Section IV, presents related works, Section V, discusses the SLA concepts and SLM meta-model. Section VI and VII describes the SLA Object and Process Model respectively. We conclude our work in the last section.

 ISBN: 978-1-61208-378-0

II. STATE OF THE ART SERVICE LEVEL MANAGEMENT SURVEY

In order to define a generic SLM framework for service delivery guarantee, it is imperative to deduce and capture important phases and process activities of SLM. Here, we present our survey derived from multiple related SLA management research works. The investigation consists of 24 SLA projects survey by the EU commission, 2 known SLA frameworks; WSLA & WS-Agreement, 2 ITSM standards; ITILv3 and CoBiT Delivery & Support, 10 individual SLA projects and 3 related SLA products in order to broaden our investigation and the applicability of SLM. Full reference list for each process activity is in TABLE I. We identified and generalized the process activities, which from our point of view, is adequately generic and sufficient for any service adaptation.

There are 4 main stages in the SLM lifecycle; 1*) Requirement Specification* – the requirement stage for service and SLA input; 2) *Instantiation and Management* – negotiation of SLA and instantiation of service and SLA; 3) *Enforcement – to* monitor and assure QoS during service runtime*;* and 4) *Conclusion* –handles the closure of service and any reimbursement of credits due to breaches in the SLA contract. There are similar works in defining these stages. In [5] defines the 5 stages while [6] shows 4 stages where these stages are categorized by its processes.

A. Requirement Specification Stage

1) SLA Template Definition - is the process of requirement capturing of the SLA contract where Service Level Objective (SLO), guaranteed state, compensation, exclusion clauses of service are defined. In ITSM practices, it includes the requirements of Operating Level Agreement (OLA), defining the service support organization structure, support contract and Underpinning Contract (UC) [7]. Output of this activity is to translate the SLA document into a machine readable format e.g., XML, ontology or rules.

B. Instantiation & Management Stage

Here we define 4 processes:-

1) *Service & SLA Offering*– is the process which advertise the *service* e.g., Virtual Server; CPU core, memory, etc; and the *service SLA attributes* e.g., QoS, performance, availability or exclusion clauses prior to the service subscription process.

2) *Negotiation* - handles the negotiation process of SLA requirements between the provider (a system providing the service) and subscriber (a system or customer). The negotiation parameter is usually limited to the qualities, which the provider is able to satisfy. The output of this activity is an agreed SLA by both parties.

3) *Mapping and translation* – In an SLO, the service parameters to be guaranteed can be in high level description, which is close to business or application requirement language. This process bridges the gap between both. The process translates the *high-level metrics* i.e., *application response time* and maps it to the *low-level resource parameter*; i.e., `transaction per-second`, `transaction response time`, `etc`. Translation includes both *functional requirements* e.g., performance, capacity and *non-functional requirements* e.g., availability, redundancy, security to be translated and mapped.

4) *Service Provisioning* – creates the actual service instance and SLA in an enforced state.

C. Enforcement Stage

Enforcement is the most important stage to ensure service delivery guarantee and service dependability during service runtime [5][8][9]. There are 5 important processes:-

1) *Monitoring* - obtains the infrastructure or application performance metrics which acts as input for the SLA's *violation detection process.*

2) *Violation detection* –monitors reactively the SLO parameters for any potential violation breach.

3) *Violation prevention* – to detect a violation before it occurs where a proactive or predictive mechanism might be use.

4) *Violation corrective action*– corrective action which is triggered by a *violation detection* or *violation prevention* process in order to repair or reduce the onset of the violation.

5) *Violation Escalation* – to escalate *error, fault* or *failure* information to system administrator where manual corrective actions can be executed.

D. Conclusion Stage

This stage consists of 5 main processes:-

1) *Termination of service & SLA* – handles the 1) proper closure due to an end of subscription; or 2) termination of service caused by a breach in the service contract.

2) *Accounting & Billing* – handles the charging mechanism to the subscriber.

3) *Resolution* – to provide remedial action or compensation of breached SLA to the subscriber.

4) *SLA template archiving* – to store previous SLA documents and its related information for future references.

5) *SLA Review* – continous review of SLA and its performance for manual SLA management as defined in ITIL v3 [7] and CoBiT [10].

We have listed out necessary SLM processes, which are deem important for the adaptation of our generic SLM framework. We however, did not list out other processes, which are too unique for a specific service implementation or explain further the derivatives of each process i.e., monitoring; dynamic monitoring, scalable monitoring or negotiation; re-negotiation or auto-negotiation processes [8].

III. AN ANALYSIS TOWARDS A GENERIC SLA FRAMEWORK

To justify our argument in requiring a generic SLA framework, we provide concrete analysis in the next few sections which discusses the dynamics of the SLM

 ISBN: 978-1-61208-378-0

management processes, SLA offering and deployment variations.

A. The Variation and Dynamics of the SLM Process

TABLE I. VARIATION OF SLM PROCESSES IN SLA IMPLEMENTATION

	EU Research	WS-Agreement	WSLA	CoBiT	ITIL v3	e-business	For SOA	A logic based	Improving Ent.	SLA@SOI	Multi-level	layered cloud	IRMOS	Cloud4SOA	OPTIMIS	WSRR	Uptime	ServiceNow
	[8]	[11]	[5]	[10]	[7]	[12]	[13]	[4]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]
Templ. Def.	X	X	X	X	X	X	X	X	X	X	X				X	X		
Offering											X							
Negotiation	X		X	X	X		X			X	X	X	X	X	X			
Map & Trans	X								X	X			X					
Serv. Provision	X	X	X	X	X			X		X	X		X	X	X			
Monitoring	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Vio. Detection	X		X			X		X		X			X	X		X	X	X
Vio. Prevention	X											X						
Vio. Corrective	X		X			X				X			X	X		X	X	X
Vio. Escalation				X	X									X			X	X
Termination			X															
Acc. & Billing	X																	
Resolution	X													X				
Archiving											X							
Review				X	X													

From the investigation done on the SLM management activities in Section II, we created a comparison matrix TABLE I, to show the variation of management process implemented in each project. For each project, we marked the processes that are being adapted. We concluded that not all of the process or activities are compulsory. For example, *negotiation*, *mapping and translation*, *accounting*, *billing*, *resolution and compensation* are some of the optional components. While *SLA template definition*, *monitoring* and *violation detection* can be considered as the *core* must-have component by deducing the total number of adaptation of the processes.

B. Types of SLA Offering

SLA offering is a service delivery guarantee commitment that the provider is willing to offer to the subscriber. The process takes place before service is instantiated. The design of the SLA offering is based on pricing strategy, customer segmentation profiling or other business factors. We conclude there are 4 types of SLA offering categories, which can affect overall implementation of the SLM process.

1) Common to all and *2) Template based SLA;* are widely used type of SLA contracts. Both types are considered as a non-negotiable contract, whereas the SLA for a particular service is fixed and applies to all or a particular segments of users. When the SLA breached, the provider will compensate by providing service credits into the customer's account or provides other forms of remedial compensation [24]. This type of SLA does not require any *negotiation* process or *mapping and translation* of monitoring metrics as the service delivery is static i.e., non-negotiable service guarantees are common to all users. It may however focus on *violation prevention* or *self-healing* capabilities in order to reduce the violation-breached effects.

3) Negotiable template based SLAs and *4) custom based SLAs* provide flexibility to the subscriber to tune the requirements to match the intended workload. Requirements example are cost, pricing, performance, availability or other attributes [16][17]. Both SLA categories require a *negotiation* process or its derivatives e.g., *auto-renegotiation* [18], *dynamic negotiation* [19][20] or *negotiation across multiple service layers* [15] i.e., between subscribers and service providers or between service providers and another service provider in a multi-level service deployment setup. It may require *mapping and translation* of high level metrics to low level metric [18][15] as the definition of SLA requirements are open for interpretation.

C. Service Deployment Variation; IaaS as a Case Study

IaaS provides infrastructural services such as compute, storage and network. Each IaaS Infrastructure deployment is unique and the deployment depends on the service functional requirements, cost, hardware, software or technology choices. It makes *monitoring* efforts; the core component of SLA system variable. We illustrate the complexity of IaaS deployment in TABLE II, which shows possible combinations of storage technology, transport, medium and backend options to host VM's virtual disk where multiple combinations can be constructed for a single deployment.

TABLE II. VIRTUAL MACHINE'S STORAGE DEPLOYMENT VARIATION

Disk	Storage Transport	Storage medium	Storage backend
File qcow2, raw, vmdk	Local	Local	Local Disk
	Over network	NFS, OCFS, GFS	JBOD, SAN
		CephFS,Gluster	Distributed Storage
Block	Over network	CEPH RBD	
	Local	LVM	JBOD, SAN
	Over network	iSCSI, AoE, FC	

We further emphasize the complexity with the example of a virtual disk deployment variation in Figure 1, where a VM can run in shared storage in *setup A* or local storage in *setup B*. Both setups require the VM Availability SLA parameter to be guaranteed but with different metrics to monitor in *Setup A*; needs to monitor host, switches, NFS storage as compared to *Setup B*, which runs on a local disk.

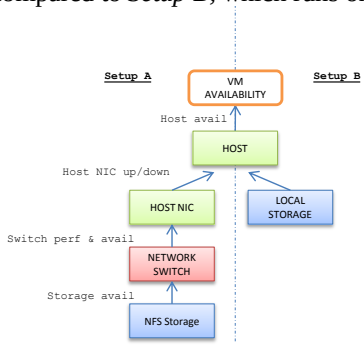

Figure 1. Deployment variation to host VM's Virtual Disk

The deployment setup will dictate the choice of performance metrics to be monitored and available corrective actions to be used. This will influence the SLA's *violation detection*, *violation prevention*, and *violation corrective action* components. In short, the service deployment will dictate the SLA *enforcement* process activities.

IaaS provides a good case study with multiple service types and multiple service deployment scenarios in order to test the concept of generic, flexible and compose-able SLA orchestrator framework.

Our investigation shows that SLM activities or processes are neither static nor absolute. It depends on multiple factors such as SLA offering types, business requirements or certain technical difficulties, which makes it harder to implement some of the components. A generic and flexible framework is needed in order to orchestrate the SLA offering by implementing all, or certain of the components towards service levels guarantee delivery.

IV. Related Work

Here, we describe two aspects of related work, the modelling of SLA and the modelling of SLM activities.

A. Data Modelling

WSLA is a framework which specifies SLAs for Web-Services implementation [25]. WS-Agreement – an OGF standard for the creation and specification of SLA [13]. SLAng model is the SLA for a spectrum of Internet Services and provides its own internal language specification [26] and lastly, a generic SLA semantic model for e-business outsourcing contract is taken as part of the survey [12]. All the above projects use UML and Object Constraint Logic (OCL) to model the SLA objects and its relationships and present it into XML schema format. In our view, this is the most suitable representation of SLA document.

Paschke, Dietrich & Kuhla [4], captures selected portion of the SLA agreement using rules and logical based Knowledge Representation concepts. Rules can be used in the function to evaluate conditions or violations in the expression for Action Guarantee. From our investigation, rule based only represents a portion of the SLA information. Service information relationships, between objects or descriptive information of the SLAs rely on other forms of representation.

Ontology based SLA model is another form of representation [14] . It captures business service performance requirements key indicators such as KPI (Key Performance Indicator and QKI (Quality Key Indicators) to define the SLA parameter. In our view, to support modularity and multiple service adaptation, domain specific knowledge should not be modelled within the same model. This hinders the generality of service implementation. For example, "availability" for software and hardware is defined differently.

Based on our literature review, we believe it is most suitable to use the UML object diagram and OCL to represent the SLA model. UML diagrams are a well-accepted software analysis and design tool. It is simple and captures a high level of information of the overall SLA document. It is sufficient to represent concepts and relationship information such as cardinality, aggregation and inheritance of those concepts.

B. SLM Modelling

There are several projects, which try to model the process flow of SLM using business process modelling. Correia & Abreu [27] proposed, a Model Driven Engineering (MDE) approach in modelling SLM activities for IT service *SLA specification* and *processes*. It uses the BPMN notation to create a process meta-model.

Corriea & Amaral [28] proposed a domain specific SLA Language Specification and Monitoring (SLALOM). It focuses on mapping of SLA to BPM notation for SLA implementation on ITIL standards. ServiceNow [23], an enterprise monitoring tool, uses a simplified form of BPMN notation for designing escalation, notification to user and scripting to automate tasks.

These are among several projects, which have the same SLM modelling objectives as ours. It provides an insight on the application and applicability of process modelling in the SLA management domain.

V. SLA Object & SLM Process Meta model

In this section, we present our deduction of SLA & SLM meta-models. We map each of SLA objects with the processes and activities of SLM to show its interactions. Figure 2 identifies the *SLA data object or concept*; presented in rectangular and *functional process;* represented in rounded rectangular. We show that, 1) *compulsory*; denoted as **(C)** – a must have activity or SLA object 2) *optional*; denoted as **(O)** - categorized as supporting activities or objects where the provider may adapt or drop.

Our SLA model extends the WSLA with additional concepts denoted as (N). In the existing WSLA model, there is no process, which uses the *exclusion* and *coverage clauses*. Both of these are being utilized by the *resolution* process. *Pricing Information* object provides information to calculate *billing and accounting*. We included the *violation* and *service action repair* object to support the *violation detection, prevention, corrective action* processes.

In the implementation of the SLA system, it is common to combine the business process flow, business logic and data model in single implementation. For example, the SLA contract rules are buried implicitly in the application code [4]. It is therefore hard to maintain the SLA when a new requirement is introduced and may require extensive re-implementation efforts. We opted the concept of addressing system complexity [29] by adapting common techniques such as abstraction and modularity.

VI. SLA Document – Data Modelling

SLA modelling is the process of synthesizing information within the SLA document and translates it into a model for the consumption of *Information Systems*. From a high-level perspective, modelling captures the service definition, objectives and guaranteed actions.

 ISBN: 978-1-61208-378-0

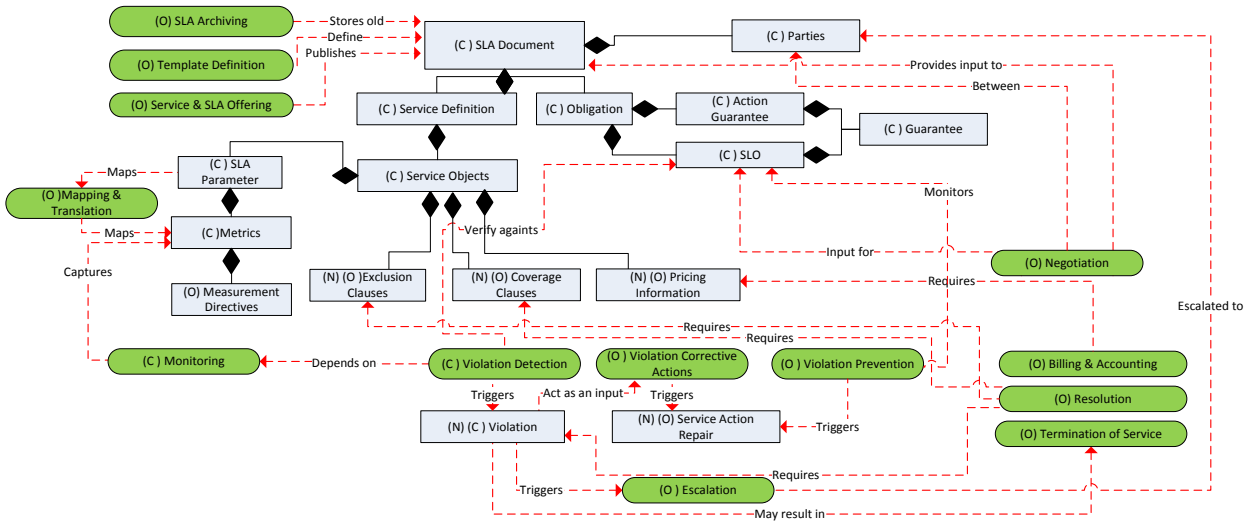

Figure 2. SLA Objects and SLM Process Meta-model

To model the SLA document requires domain experts in depth understanding of the ITSM practices. Thus instead of creating a new model, we opted an existing one. We chose an SLA model that is generic enough to be able to describe multiple services implementation and a model that is non-specific to any service implementation. We opted the WSLA as our case study.

The WSLA describe an SLA framework for web-service implementation. It is generic enough to be adapted by multiple service categories, such as service management, networks or business application. The data model captures the crucial attributes, which is used to measure and monitor the QoS parameters, violation detection, repair actions logics, and escalation mechanism to authorized parties [25]. The WSLA framework provides a layout of a SLA document schema and provides multiple custom type object definitions. The WSLA consists of 3 major parts: 1) *Service Definition* – captures the service definition, the SLA parameter to guarantee and metrics to monitor 2) *Obligation* – captures the SLO and action guarantees towards the state or the violation, and 3) *Parties involved* - between the signatory or supporting party that supports SLM.

To use and implement the WSLA introduces a new set of challenges in the creation, modification and management of the SLA template. WSLA does not separate the 1) *service objects definition description and relationship*; with 2) *rules, logics and algorithm* and 3) *runtime* and *implementation information*. Thus, the ***reusability*** – using the *parts-of* the SLA model and ***portability*** – to be able to export or import parts-of the SLA model is not possible. For example, in WSLA SLAParameter it captures the "how to measure", "how to aggregate" into a *compositeMetrics* and includes, which "party" is responsible to provide the metric value. From our point of view, this information is only unique for a particular service implementation and mixing the modelling information with the implementation information will hinder reusability & portability.

Based on these challenges we segmentize the SLA model and decouple with clear natural separation boundary by adapting the Separation of Concern, a design technique to achieve modularity as being implemented in [4][15][16][30]. Modularity promotes reusability and portability parts-of the SLA model i.e., by allowing the provider to reuse or imports parts of the SLA model for new service implementation. This is to satisfy and support multiple implementation of the service based on the challenges defined in *Section I.*

We suggest 4 separations of the SLA model:-

1) **WHAT-IS** *the SLA-Model* ***(SLA-M)*** – *information about the* `ServiceDefinition`, `ServiceObject`, `SLAParameter` *and its relationship;*

2) **WHAT-IF** the SLA-Logics **(SLA-L)** – the logics and rules of `SLO`, `actionGuarantee`, and *monitoring calculation.* It may consist of `measurementDirectives` expression which calculates composite metrics i.e., high-level metrics such as availability may consist of multiple low-level metrics aggregation. Both can be defined using a combination of `function` and `expression`;

3) **HOW-TO** the SLA Implementation (**SLA-I**) – the service implementation information such as *Obligation-action*, monitoring GET `metricURI` endpoints, action SET `actionURI` endpoints together with;

4) **THE RUNTIME (SLA-R)** - the SLA information, which is populated during service and SLA runtime. For example, signatory parties information, and possibly the QoS parameters.

The SLA-M can be considered as a SLA service catalogue, which defines (`ServiceDefinition`) and quality attributes to guarantee (`serviceObjects`) and SLA Parameter to monitor (`SLAParameter`) the provider guarantees. It is considered as a non-volatile SLA data model where it is rarely changed unless the provider can guarantee a new `serviceObject`. We explain further based on, extends the SLA-M `ServiceObject` type. `ServiceDefinition` is a Virtual Machine service,

where the serviceObjects may consist of *VMAvailability*, *VMBootTime* and *VMNetworkPerformance*. For each of the ServiceObject, the provider has agreed to guarantee these qualities. By guarantying these qualities, we assume the providers have tested, able to collect the monitoring metrics and probably have a corrective action plan in hand to maintain the SLA for these qualities. If new ServiceObject is introduced, then a new sets of SLAParameter, metrics, SLO and ActionGuarantee needs to be laid out.

TABLE III. TIERING WSLA EXAMPLE

Type	WSLA Elements	Examples
SLA-R	**ServiceProvider**	
	Name	MIMOS cloud service
	Role	Provider
	Contact	Kuala Lumpur, MY
	Action	n/a
SLA-D	**ServiceDefinition**	
	Name	Virtual Machine Service
	Description	Virtualize infrastructure
SLA-D	**ServiceObject**	
	Name	VMAvailability
	Desc	Virtualization Availability
	Schedule	24X7 Coverage
	Trigger	On Service Hooks
	Constant	99.99%
SLA-D	**SLAParameter**	
	Name	VMAvailabilityUptimeRatio
	Unit	%
	Type	FLOAT
	Category	Availability
SLA-I	**Metric 1**	
	Name/Type/Unit/Function	VMStatus / INT / Boolean /N/A
SLA-I	**Metric 2**	
	Name/Type/Unit/	VMUptimeRatio / LONG / % /
SLA-L	Function	If(VMStatus==0) Then VMUptimeRatio-1%
SLA-R	SLO	
	Name	VMUptimeSLO
	ValidityStart	01:01:00 2014/06/01
	ValidityEnd	When Subscription Ends
	Expression	PREDICATE= GREATERTHAN
		SLAParameter >Name =
SLA-L		VMAvailabilityUptimeRatio, Value=0.95

The SLA-I is the require information in the *implementation* phases. For example, GET – to fetch the monitoring data information or service status, and SET – to react based on the violation status. All of the above depends on the deployment setup landscape. This information is rarely changed unless the same SLA-I model is applied to another service deployment setup. The SLA-I captures time or interval information such as schedule starts and stops, period e.g., *"SLO is valid only on working days"*, interval of monitoring data to be fetch or send notification response in intervals.

The SLA-L captures the rules, and logic. In WSLA, it can be in a form of evaluation expression or function in SLO, or ActionGuarantee *e.g., "live migrate the VM if underlying HW components fail"*. A sample SLA-L is SLO expression, for example *VMAvailabilityUptimeRatio >= 0.95* as depicted in table above. SLA-L can be depicted in 2 forms; 1) monitoring logics or violation detection; 2) violation prevention or violation correction action logic. The logic can change depending on the requirements, towards the service. For example, changing how the availability formula calculation or modify the action to perform once a violation is detected will affect the SLA-L.

SLA-R is the *SLA runtime* information during instantiation and enforcement stages. It records information captured during SLA instantiation and the enforcement lifecycle. In Negotiation process such as contractual information; parties involves in the SLA agreement, agreement date, and agreement validity period are captured into the SLA-R.

VII. SERVICE LEVEL MANAGEMENT – PROCESS MODELLING

Business Process Management (BPM) transforms real world business processes into a *process model representation*. Several standards are available, for example BPMN, EEML, Flowchart, BPEL and IDEF3. Business Process Model & Notation (BPMN) is one of the most widely accepted process modelling standards. It creates a standardized bridge between business process design and process implementation. To design a process, Business Process Diagram (BPD) provides a flowcharting technique to create graphical models of the process called workflow. The same workflow, with enough customization and coding, can be executed by any BPM System (BPMS) e.g., Activiti, Bonita, jBPM. Thus, it reduces the gap between analysis, design and implementation of the system.

Due to the nature of dynamics of SLM process varieties, we need an approach, which could address the design, analysis and implementation challenges. We propose to use BPMN, and utilize the BPD to model the SLM. To model, we need to identify the SLA concept, process flow and interactions of system component, required state, triggering events etc. It will act as a guideline for implementing the SLM into BPMN.

In delivering the SLA, SLM activities manage the SLA states throughout *creation*, *instantiation*, *enforcement* and *termination*. The provider needs to orchestrate the SLM activities. Here, we try to explain those processes and see its applicability throughout the state of service and SLA (TABLE IV).

TABLE IV. SLA, SLM, SERVICE AND T-WSLA STATE AND ACTIVITIES

	t1	t2	t3	t4	t5	t6
Service State				Instantiation	Runtime	Terminate
Service Activities		Service Offering		Service deployment		Service termination
SLA & SLM State	SLA, SLM creation	SLM Instantiation		SLA Instantiation	SLA enforced	SLA & WF SLM terminated, SLA Archiving
SLM Activities	Template definition		SLA Negotiation process	SLA Deployment, Mapping & Translation	Evaluation & Enforcement, monitoring	Conclusion - Accounting, Settlement/Penalties/reward
T-WSLA State	Define SLA-D, SLA-I and SLA-L	Display the SLA-M info to the customer; Service Definition, SLAParameter and exclusion.	Capture the SLA-R Signatory information between parties		The SLA-L and SLA-I will be used throughout the service runtime state	

 ISBN: 978-1-61208-378-0

At t0, the assumption is made that the service is ready to serve the customer. In t1, the *creation* of SLA data model (SLA-D, SLA-I, SLA-L) and SLM process model is formalized by the provider. This is the design and creation state of T-WSLA Object & SLM process workflow. At t2, the service & SLA agreement is *offered* to the user and the SLM workflow starts in parallel. In t3, the SLA enters the negotiation process. The negotiation process is not compulsory and depends on the design of the SLM. Once negotiation completes, we populate contractual information into the SLA-R. In t4, the service and SLA is instantiated. In t5, the Service is *running* and the *SLA* is in *enforced mode* where continuous evaluation, enforcement and monitoring are executed. The SLA-Logic that includes the SLO, goals, metric calculation and SLA-I i.e., monitoring URI and action URI is continuously used in t5. At T6, when the service is terminated, the SLA & SLM activities will be stopped. The SLA conclusion processes like accounting, settlement, penalties or rewards may run in both t5 and t6 and not exclusively in t6 since penalties can be compensated even during service runtime.

A. Conceptual Modelling of SLA to BPMN

TABLE V. CONCEPTUAL MAPPING OF SLA TO BPMN NOTATION

Concept	SLA Entity	BPMN
To show the data object transition between processes	T-WSLA	Data Object
To visualize signatory parties or communication between systems or components.	Role & Organization	Lanes & Pool
To express individual SLM process	negotiation, enforcement, service offering, conclusion	Sub-process; Process elements; tasks, gateway, events
To calculate composite metrics or SLA parameter	Monitoring metrics, SLA Parameters	
To express complex violation logics	Violation detection, prevention, action	
ActionGuarantee expresses a commitment to perform action if a given precondition is met.	Action Guarantee	
SLO expresses commitment to maintain a particular state of the service over a period of time.	SLO logics	Sub-process for complex representation of SLO and business rules
Any rules expressed in the SLA document.	SLA Exclusion, Termination clauses, penalties	Business rules or conditional Events
To escalate message, system signal to another party or system component.	Escalation events	Intermediate throwing or end (escalation, message, signal) events
To receive violation events.	Violation events	Start or intermediate catching (error, message, signal) events
Scheduling of SLA or Monitoring.	Schedule	Timer Event
SLA period or others.	Period	Timer Event
To depict interval i.e., monitoring collection intervals.	Intervals	Timer Event

There are 4 basic categories of notations, 1) *flow object* (*Event, Activity, Gateway*) –core objects types to represent the operation logics. 2) *Connecting objects* (*sequence, message, association flow*) – to show communication or interaction, 3) *swimlanes* (pools, lane) – to illustrate different functional capabilities or responsibilities. And 4) *artifacts (Data Objects, Group, Annotation)*- as a supplementary notation [31].

A conceptual mapping is discussed in TABLE V between the SLA [25] to BPMN 2.0 specification [32]. This provides a guideline on how to use the BPMN in modelling the SLM process is possibly incomplete or accurate and open for improvement. To illustrate further, we can express the SLA violation triggers from monitoring tool or workflow sub-process as start or intermediate catching; error, message or signal events or escalation of events to parties can be formalize using intermediate throwing or end; escalation, message or signal events. Any form of rules for example, SLA exclusion, and terminations can be put into complex business rules or simplified conditional events at process flow gateways.

B. Example of IaaS Service into BPM Notation

To illustrate BPMN notation, we present the VM availability *Service Object* into BPMN notation in Figure 3. It shows the *a) Enforcement stage* and b) *conclusion stage*; processes. This process model facilitates the SLA specification in the design phase of SLM activities as well as the interpretation of events during SLA monitoring.

BPMN, similar to BPEL deals with two parts of process modelling 1) the *abstract* – partially specified processes that are not intended to be executed, which hides some of the required operational details of a process. Abstract processes serve as a descriptive role with more than one possible use case, which includes the behaviour or events of the process i.e. the communication, function or states of processes. It acts as a management discipline where it was used originally for people-to-people communication through BPD modelling. The process may highly underspecify where the process is presented in high-level descriptions. 2) The *executable business process* is an actual behaviour of a participant in business interactions which is executed in a BPM-system (BPMS) such as Activiti or jBPM [33].

While this paper presented the SLM into the BPMN abstract process design model, it is imperative to mention that the implementation to deliver SLA will requires more programming effort in order for it to run in a BPMS workflow engine.

We have modelled the SLM and monitoring logics into BPMN, which can be executed on workflow engines. However, to reduce the traffic and monitoring overhead, it is best to push and translate the workflow into monitoring specific implementation through bash scripts or any available monitoring scripting available.

 ISBN: 978-1-61208-378-0

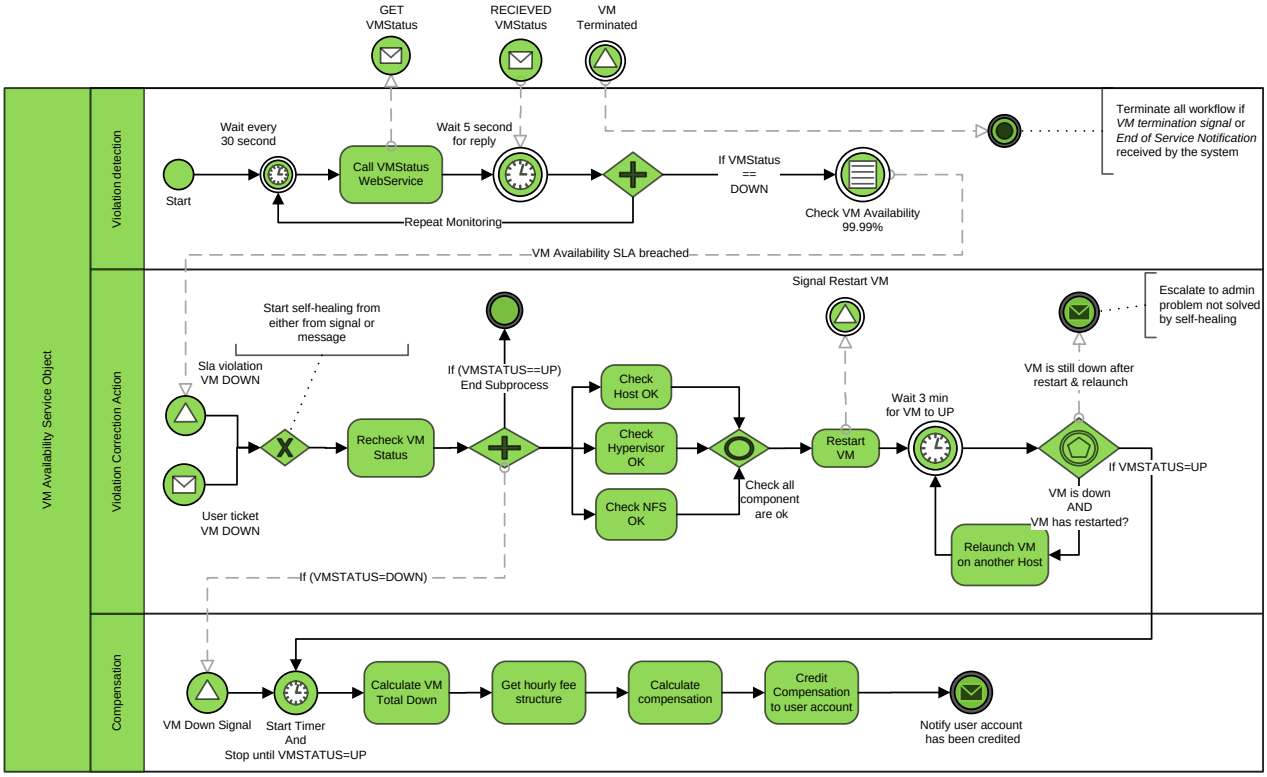

Figure 3. Enforcement & compensation BPMN Notation

VIII. CONCLUSION & FUTURE WORK

In this paper, we have summarized relevant SLA objects and SLA management processes towards a generic SLA management framework. To design a good SLA orchestrator tool it is imperative to properly 1) represent the SLA document into a suitable SLA model and 2) SLM management process into a suitable process language notation. Based on our survey, we have derived the SLA and SLM meta-model to show the relationship, interaction and categorize those items as compulsory or optional. In designing the SLA offering and its management process, the framework should be flexible enough to incorporate or drop any of the process or object as the SLA offering is never a fixed process cycle.

To model the SLA, we opted for WSLA and extended its concepts in order to support new SLA process activities. We then proposed a tiered mechanism within the WSLA, which separates the information, logics, implementation and runtime information to promote reusability parts-of the model. We illustrate the usage with an example of an IaaS VM availability scenario.

To model the processes, we provide a concrete conceptual mapping of SLA objects and concepts to the notation of BPMN. In our opinion, the BPMN is the most suitable process modelling language in order to design and implement SLA. The BPMN captures the nature, variation of design and dynamics of the SLM processes. We believe that by correctly defining those two models it will act as a guideline in creating the SLA offering.

For future work, a proof of concept of the framework is required. The tool should be flexible in defining some or a combination of the SLA management process. IaaS would be a suitable test case as it is a complex and ever-changing technology.

REFERENCES

[1] J. Happe, W. Theilmann, A. Edmonds, and K. T. Kearney, “Service Level Agreements for Multi-Level SLA Management,” Service Level Agreements for Cloud Computing, P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds. New York, NY: Springer New York, pp. 13–26, 2011.

[2] R. Sturm, W. Morris and M. Jander, “Foundations of Service Level Management.” Indianapolis: SAMS Publisher, p. 272, 2000.

[3] I. Rosenberg, A. Conguista, and R. Kuebert, “Management for Service Level Agreements,” Service Oriented Infrastructures And Cloud Service Platforms For The Enterprise, T. Dimitrakos, J. Martrat, and S. Wesner, Eds. Springer Berlin Heidelberg, pp. 103–124, 2010.

[4] A. Paschke, J. Dietrich, and K. Kuhla, “A logic based sla management framework,” in Semantic Web and Policy Workshop (SWPW) at 4th Semantic Web Conference (ISWC 2005), 2005.

[5] A. Keller and H. Ludwig, “The WSLA framework: Specifying and monitoring service level agreements for web services,” J. Netw. Syst. Manag., vol. 11, no. 1, pp. 57–81, 2003.

[6] X. Liu, Y. Yang, D. Yuan, G. Zhang, W. Li, and D. Cao, “A Generic QoS Framework for Cloud Workflow Systems,” 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 713–720, 2011,.

 ISBN: 978-1-61208-378-0

[7] OSIATIS S.A, "Service Level Management - Introduction and Objectives," 2014. [Online]. Available from: http://itil.osiatis.es. 06-Apr-2014
[8] D. Kyriazis, "Cloud Computing Service Level Agreements;Exploitation of Research Results," European Commision Directorate General Communications Networks, Content and Technology, Technical Report, 2013.
[9] V. C. Emeakaroha, I. Brandic, M. Maurer and S. Dustdar, "Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," The 2010 High Performance Computing and Simulation Conference (HPCS 2010), pp. 48–54, 2010.
[10] ISACA, "Information Systems Audit and Control Association." [Online]. Available from: http://www.isaca.org/. 23-Oct-2014
[11] "WSAG4J Programming Model." [Online]. Available from: http://wsag4j.sourceforge.net/site/server/programming_model.html. 06-Apr-2014
[12] C. Ward, M. J. Buco, R. N. Chang, and L. Z. Luan, "A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts," E-Commerce and Web Technologies, pp. 363–376, 2002.
[13] H. Ludwig, "WS-Agreement Concepts and Use–Agreement-Based Service-Oriented Architectures," IBM Research Division, Technical Report, 2006.
[14] A. Asosheh and P. Hajinazari, "Towards improving enterprise performance with Service Level Agreements," Telecommunications (IST), 2012 Sixth International Symposium, pp. 913–918, 2012.
[15] W. Theilmann, J. Lambea, F. Brosch, S. Guinea, P. Chronz, F. Torelli, J. Kennedy, M. Nolan, G. Zacco, G. Spanoudakis, M. Stopar and G. Armellin, "SLA@SOI Final Report," European Commission Information Society and Media, Technical Report, 2011.
[16] W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, and J. Lambea, "A reference architecture for multi-level sla management," Journal of Internet Engineering, vol. 4, no. 1, pp. 289–298, 2010.
[17] I. Haq, I. Brandic, and E. Schikuta, "SLA Validation in Layered Cloud Infrastructures," Economics of Grids, Clouds, Systems, and Services, R. Y. Philipp Wieder, Joe M. Butler, Wolfgang Theilmann, Ed. Springer Berlin Heidelberg, pp. 153–164, 2010.
[18] "IRMOS Project." [Online]. Available from: http://www.irmosproject.eu/. 23-Oct-2014
[19] "Cloud4SOA Project." [Online]. Available from: http://www.cloud4soa.eu/. 23-Oct-2014
[20] "OPTIMIS Project." [Online]. Available from: http://www.optimis-project.eu. 23-Oct-2014
[21] M. Smithson, "Applying SOA Governance Using WSRR, DataPower, and WS-Policy." [Online]. Available from: https://www-950.ibm.com/events/wwe/grp/grp004.nsf/vLookupPDFs/Applying SOA Governance Using WSRR, DataPower, and WS-Policy/$file/Applying SOA Governance Using WSRR, DataPower, and WS-Policy.pdf. 23-Oct-2014
[22] "Uptime Software." [Online]. Available from: http://www.uptimesoftware.com/. 20-Jun-2014
[23] Servicenow, "Service Level Agreements." [Online]. Available from: https://wiki.servicenow.com/index.php?title=Service_Level_Agreements_(SLA)_Plugin. 01-Jun-2014
[24] J. Meegan, G. Singh, S. Woodward, S. Venticinque, M. Rak, D. Harris, G. Murray, B. D. Martino, Y. L. Roux, J. McDonald, R. Kean, M. Edwards, D. Russell and G. Malekkos, "Practical Guide to Cloud Service Level Agreements version 1.0," Cloud Standard Consumer Council, Technical Whitepaper, 2012.
[25] H. Ludwig, A. Keller, A. Dan, R. P. King and R. Franck, "Web service level agreement (WSLA) language specification," IBM Corporation, Technical Report, 2003.
[26] J. Skene, D. D. Lamanna, and W. Emmerich, "Precise service level agreements," International Conference on Software Engineering (ICSE 2004), pp. 179 – 188, 2004.
[27] A. Correia and F. B. e Abreu, "Model-driven service level management," 4th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2010), pp. 85–88, 2010.
[28] A. Correia, F. B. e Abreu and V. Amaral "SLALOM: a Language for SLA Specification and Monitoring," Computing Research Repository (CoRR 2011), pp. 556–567, 2011.
[29] J. Saltzer and M. Kaashoek, "Principles of computer system design: an introduction." Morgan Kaufmann, 2009.
[30] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France, "Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development," 29th Annu. Int. Comput. Softw. Appl. Conf., vol. 1, pp. 121–126, 2005.
[31] S. A. White, "Introduction to BPMN," IBM Cooperation, Technical Paper, pp. 1–11.
[32] "BPMN 2.0 by Example-version 1.0," Object Management Group, Technical Paper, 2010.
[33] Acitiviti, "What is BPM." [Online]. Available from: http://activiti.org/faq.html#WhatIsBpm. 07-Mar-2014

 ISBN: 978-1-61208-378-0

HiPAS: High Performance Adaptive Schema Migration

Evaluation of a Self-Optimizing Database Migration

Hendrik Müller, Andreas Prusch, Steffan Agel
Pasolfora GmbH
An der Leiten 37, 91177 Thalmässing, Germany
{hendrik.mueller|andreas.prusch|steffan.agel}@pasolfora.com

Abstract – HiPAS is a database migration method, aimed at reducing downtime during offline migrations by automatically adapting to available system resources. Investigating the applicability of adaptive capabilities for database migrations, two stages of system complexity, adaption and anticipation, were mapped onto the requirement of utilizing a system up to an optimal degree in order to achieve the shortest possible transfer duration. The developed method is automated by implementing the HiPAS software, which adapts to its environment by continuously monitoring relevant system information, and increasing or decreasing the current parallelization degree whenever necessity is assumed. To enable a flexible adaption, the total amount of migration data is partitioned into equal sized transfer jobs being distributed across available instances and networks. Since HiPAS is invoked on the database layer, and controlled by a temporarily created autonomous database user, migration metadata is stored inside tables thus being highly integrated with the actual migration data. HiPAS was designed and evaluated iteratively following the IS research framework and reveals significant downtime reduction potential compared to non-adaptive migration approaches like Oracle "Data Pump". Our results serve as a contribution to all researchers and practitioners in investigating fields of application for adaptability mechanisms.

Keywords-Adaptability; Anticipation; Database Migration; Parallelization.

I. INTRODUCTION

The rapid technical developments inside changing markets, as well as the need for efficiency enhancements, mainly driven by cost pressure, require to transfer running information systems occasionally into a new environment, which fulfills the operational requirements in a more suitable way. This process is referred to as software migration [1] and meanwhile the software's availability can be limited depending on the chosen migration method. Regarding this, basically two approaches can be differentiated:

- online Migration: continuous availability
- offline Migration: interrupted availability

In some critical environments, a downtime is not acceptable, thus online migrations need to be performed. This paper deals with the variety of cases, which do not require a costly and complex online migration and a planned downtime is tenable. In that case the main concern is to keep the downtime as small as possible since the duration of unavailability may result in opportunity costs. In particular, we target migrations applying the "big-bang" strategy [2], thus data is fully migrated at once in contrast to incremental migrations. Since the legacy system (source system) is shut down during the data transfer, starting the target system, referred to as cut-over [3], cannot be performed before all required data has been transferred to the target system's database. The length of downtime depends on the migration approach taken. For database migrations, different system layers can be involved determining the performance and granularity of data selection (see Section 2). We investigated the applicability of adaptive capabilities for database migration software in order to reduce the necessary downtime by parallelizing data transfer up to an optimal parallelization degree, which will be continuously adapted to the system's load capacity. Prior tests indicated, that overloading the target or source systems resources leads to a temporary stagnation of the whole migration progress, whereas a low utilization wastes available resources, thus underachieving existing downtime reduction potential.

The developed approach "HiPAS" (High Performance Adaptive Schema Migration) is intended to provide dependability and interruptibility, since migration software should be able to identify where to resume an interrupted migration process instead of starting from scratch avoiding the necessity of rescheduling a planned downtime.

Further technically conditioned features will be added in Section 3 as consequences of the preliminary considerations. Section 4 summarizes HiPAS´ architecture by means of introducing adaptability challenges of the subsequent described migration process. The adaptive capabilities are outlined in Section 6 and 7. Finally, in Section 8, we evaluate HiPAS, which refers to both the designed migration method and the migration software, currently implemented in Oracle PL/SQL syntax comprising 8,540 lines of source code.

II. PRESENT MIGRATION APPROACHES

As introduced previously, migration approaches can be differentiated regarding the availability of the migrated systems into online and offline migrations. For stated reasons, we focus offline migrations, which can be further classified concerning their own characteristics and their applicability for certain database characteristics:

- invocation layer
- support for change of platform

 ISBN: 978-1-61208-378-0

- support for change of endianness
- support for change of character set
- downtime proportionality

The divergence of the source and target database in terms of platform, endianness and character set technically limits the available migration methods. A critical decision criterion for the remaining contemplable methods is the demand for downtime shortness resulting in lower opportunity costs during the unavailability of the database and all relying applications. The fact, that a high throughput for data transfer was achieved as yet by eliminating upper layers and protocols, leads to the conflicting goals of flexibility and performance when selecting a migration method. The lower a layer a migration is invoked on the more flexibility is lost, since changes of database characteristics might not be supported and the possible granularity for migration data selection decreases. Finally, downtime proportionality refers to the entity, which the downtime length depends on; this can be the amount of migration data or the data alteration rate if incremental methods are used.

When designing HiPAS, we pursued the goal of achieving a short downtime and at the same time providing the flexibility of migrating between divergent databases and selecting the data as granular as possible. This was achieved by invoking the migration on database layer without ever leaving this layer during the whole migration process and by parallelizing the data transfer adaptively in respect of the system's resources. Therefore, we add "adaptability" as a further decision criterion for migration software capabilities.

III. PRELIMINARY CONSIDERATIONS

The performance of migration software highly depends on how well its design fits to the operating environment and the intended range of functions. Previous system and data analyses are necessary to conclude with a migration design, which has been aligned to the findings in multiple iterations following the guidelines of design science in information system research [4]. Figure 1 shows, how the designed artefact HiPAS is related to its environment and knowledgebase base inside the information systems research framework.

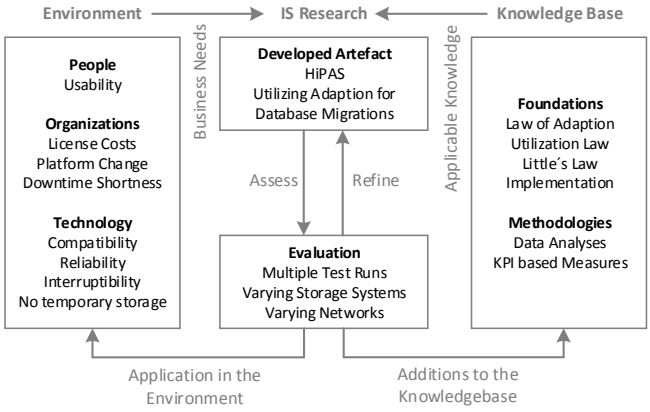

Figure 1. HiPAS as an IS Research Artefact (Adapted from Information Systems Research Framework [2]).

HiPAS was intended to be built upon findings of preliminary analyses (Knowledgebase) described in this Section as well as business requirements (Environment) and from then on has been improved continuously, based on evaluation runs performed in a variety of different environments provided generously by customers.

A. Enterprise Data Structures

When moving existing data files to the target system, as migration approaches invoked on storage and database layer do (see Section 2), the valuable downtime is partly spent migrating unnecessary or useless data. The allocated size of a data file implies unused space and indexes. To gain an overview of typical storage occupancies, we analyzed 41 SAP systems productively running at a German public authority by querying the allocated disk space, the used disk space and the space used for indexes with a result shown in Figure 2.

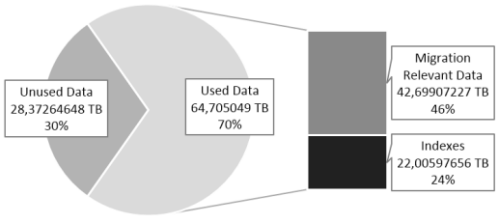

Figure 2. Average Structure of Allocated Data.

For these 41 SAP Systems, we identified an overall amount of 93.08 TB allocated data. From this amount, about 28 TB (30 %) represented allocated space, which was not yet filled with data. From the used space of 65 TB, about 22 TB (24% of the overall amount) were filled with indexes. The remaining 43 TB (46% of the overall amount) represent the actual relevant data, which necessarily needs to be transferred into the target database within a migration. Indexes can be created at the target system and do not have to be transferred, thus saving network bandwidth. Depending on the layer the migration is invoked on, unused but allocated data can be excluded as well.

In this case, if all of the analyzed SAP systems needed to be migrated, migration tools not supporting data selection would utilize all involved system resources for transferring data, of which approximately 54% is useless on the target system. Invoking a migration method on software layer enables both excluding useless data autonomously and implementing self-adaptability.

B. Endianness

When performing a database migration, the byte order in which the source and target system store bytes into memory needs to be considered. This byte order is referred to as endianness and data is stored into data files accordingly, so the endianness can affect the amount of available migration methods and the overall needed downtime.

A major part of migration demanding customers served by the authors of this paper currently initiate migration projects due to licensing and maintenance costs, this amount

is strongly influenced by an increasing number of platform migrations from Solaris to Linux, requiring subsequent migrations on upper layers such as the databases tier. The latest International Data Corporation (IDC) report on worldwide server market revenues substantiates this observation by stating, that Linux server revenue raised from 17% in Q4 2010 to 23.2% in Q2 2013 compared to Unix decreasing from 25.6% down to 15.1% [5]. The Unix-based Solaris operates on processors following Oracle´s SPARC architecture, whereas Linux distributions can be used on systems based on Intel processors. When migrating from Solaris to Linux accordingly the endianness changes from big endian to little endian, so the data files cannot simply be moved without converting them before or after the transfer.

Alternatively to converting data files, the database migration can be invoked on a layer, which supports saving the data into new files on the target system such as export-import-tools as well as HiPAS do. In this case, migration performance can be enhanced by means of adaptive capabilities.

C. Storage I/O Controller

As a consequence of the requirement for downtimes as short as possible, a utilization degree of the underlying storage systems has to be achieved, which enables short response times. The overall amount of requests inside a system (N) equals the product of arrival rate (a) and average response time (R) as expressed by Little´s Law [6]:

$$N = a \times R \tag{1}$$

In addition the Utilization Law [7] defines the utilization (U) of the I/O controller as the product of arrival rate and average processing time (R_S):

$$U = a \times R_S \tag{2}$$

By combining these relations, it becomes clear that the response time depends on the I/O controller's utilization as described within the following formula: [8]

$$R = \frac{R_S}{1 - U} \tag{3}$$

The relation shows, that the response time does not change linearly to the utilization. At higher utilizations, the response time grows exponentially as clarified in Figure 3.

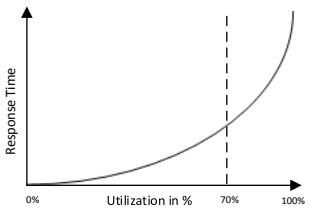

Figure 3. Relation of Utilization and Response Time.

By adapting to the source and target system resources, HiPAS continuously changes the utilization of the I/O controller in order to achieve an optimal relation of response time and utilization supporting the shortest possible overall duration. The storage manufacturer EMC generally describes an average utilization of 70% as optimal [8].

IV. HiPAS Architecture

Following the goals introduced in Section 1, we designed the HiPAS migration method as describes in the following.

A. Everything is a Tuple

When performing an automated and controllable migration, a number of interim results arise, e.g., during the analysis of source data. Keeping these information as well as logging and status information is necessary for the administrator to manage and verify the migration and for the software itself to handle parallel job executions autonomously. The necessity for saving and querying migration metadata leads to HiPAS's design paradigm of not leaving the database layer during the whole migration process. Interim results such as generated DDL and DML Statements for later execution are represented by tuples of tables inside a temporary migration schema enjoying advantages of the databases transactional control mechanisms. The paradigm of everything being a tuple is emphasized by the following list:

- objects to create are tuples (table "cr_sql")
- data to transfer are tuples (table "transfer_job_list")
- running jobs are tuples (table "mig_control")
- parameters are tuples (table "param")
- logs are tuples (table "logging")

After a migration has been performed, its success and the transferred data´s integrity have to be verified. Since logging information was stored during the whole process inside the logging table, SQL can be leveraged to query for certain transferred objects or states or both. Sorting, calculating and analytical capabilities of SQL are utilized as well for optimizing the migration process, thus there is no need for any other migration application on operating system level then the database management system (DBMS) itself.

B. Adaptability and Dependability Problems

When designing the migration method and implementing the related software, several challenges had to be faced. In this Section, we will briefly introduce some of the most interesting problems and their intended solutions:

- Utilization Problem
- Knapsack Problem
- Distribution Problem
- Dependency Problem
- Index Problem

Subsequently described solution approaches for the above listed problems will provide an overview of the conceived migration method. In-depth Sections are referenced.

1) Utilization Problem: Utilization cannot be planned generally since systems behave differently depending on

their resources and further running processes. During the evaluation phase performed migration test runs having a preliminary defined static parallelization degree, verify this statement, which leads to the risk of both overloading a system and on the other, hand not utilizing idle resources. Derived from the relationship between utilization and response time described in Section 3-C, Figure 4 shows how the overall performance, in terms of response time, behaves at increasing parallelization degrees:

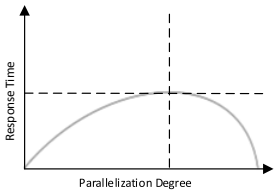

Figure 4. Expansion when Overloading the Storage System.

By choosing the currently optimal parallelization degree adaptively at any time, HiPAS targets an optimal and dynamic utilization and, in this way, reduces the risk of utilizing the systems too much or not enough. Parallelization is implemented by means of background jobs started through the database scheduler. In this way, the yet manual task of finding the optimal parallelization degree for the respective system environment is intended to be done by HiPAS automatically and adaptively, implicating the ability to change this value dynamically during the whole transfer process.

2) Knapsack Problem: From an amount of objects, defined by their weights and values, a subset with limited weight and maximum total value has to be chosen [9]. This knapsack problem reflects the challenge of choosing optimal combinations of different sized tables to transfer in parallel, since the available computing resources are limited. Large tables should be preferred in a way of starting their transfer at the beginning of the migration process, because a possible failure can require a restart of the table transfer thus delaying the whole migration when started too late. HiPAS circumvents the knapsack problem by dividing large tables into equal sized partitions, which can be transferred in parallel. This offers flexibility in scheduling the data transfer and dynamically adapting the current parallelization degree.

3) Distribution Problem: Depending on the migration environment, the accruing work load can be distributed on multiple instances of a cluster. In terms of network bandwidth, multiple database links can be created on different physical network connections between the source and target system. In this case, HiPAS will distribute data to be transferred equally on the available database links in order to utilize the total available bandwidths. In case of a real application cluster (RAC), HiPAS distributes running transfer jobs on the available instances. Then the fact of the previously mentioned partitioning of large tables needs to be considered. We optimized the data buffers of the instances by distributing transfer jobs, which continue a large table, to the instance, which already transferred previous parts of the same table to avoid reloading the table into multiple buffers of different instances. The corresponding algorithm is explained in Section 7-D.

4) Dependency Problem: When invoking the migration on database layer, dependencies among the transferred objects need to be considered for the transfer order. Surely, users need to exist before importing data into created tables and granting permissions found in the source schema. Constraints like foreign keys have to be disabled temporary, so HiPAS does not have to spend time for calculate a strict and inflexible transfer order. If reference partitioning was used inside the source schema, a parent table needs to exist before the child table can be created following the same partitions. For considering such dependencies, HiPAS calculates a transfer schedule in the first place. Since possible existing triggers will be transferred as well, they need to be disabled during the migration process in order to avoid unexpected operations on the target system, e.g., invoked by an insert trigger.

5) Index Problem: Indexes can either be created directly after table creation or after the table has been filled with data. When creating the index before data load, they will be built "on the fly" during the transfer phase, in contrast, after data load, an additional index buildup phase would need to be scheduled. The right time for indexing depends on the target storage system and network bandwidths. In case of a highly powerful storage system, it might be reasonable to build the indexes directly during data import since the network represents the bottleneck of the whole migration and the storage system would idle otherwise. On the other hand, storage systems can be overloaded when indexes have to be created at import time. Consequently, the decision about the indexing time is another use case for the adaptive capabilities of HiPAS explained in Section 7-C.

V. COMPONENTS AND MIGRATION PROCESS

Assuming, that both source and target database system have been physically connected preliminary and are configured to be accessible by each other, the migration process consists of three main phases invoked on the target system, which are briefly described subsequently:

1. Installation and Pre-Transfer (Step 1-3)
2. Adaptive Data Transfer (Step 4-6)
3. Post-Transfer and Uninstallation (Step 7)

Figure 6 on the next page shows the steps of these phases, which are invoked on the target system.

A. Installation and Pre-Transfer

Following the paradigm of not leaving the database layer, an additional and temporary schema is created inside both source and target database during an automated installation phase. All subsequent operations will be done by the owner

of this schema. Creating this user as well as creating and compiling a PL/SQL package, needed for performing the migration, is part of an automated installation process. Prior to the data transfer phase, the source schemas need to be analyzed and accordingly created inside the target database. For this purpose, SQL statements for creating the identified objects will be generated and stored inside the table "cr_sql_remote". This table will be copied to the target site and contains information regarding the objects to be created and its creation status. In addition, every operation performed causes status information to be written into the table "logging" (see Figure 5), enabling the database administrator to perform any necessary analysis, e.g., by querying for possible errors during or after the migration:

```
select logdate, loginfo from logging where info_level
= 'ERROR';
```

After the initial analyses of the source schema, all identified objects have the status "init" and will therefore be created by HiPAS at the target site. All objects containing "created" inside their corresponding status column will be ignored, enabling the whole migration process to be paused and continued at any time. The table "param" (see Figure 5) serves as a user interface for parameterizing HiPAS manually beforehand, in case certain adaptive capabilities shall not be utilized.

Techniques like reference partitioning inside the source schema have to be considered and will determine the order of creation, since child tables will not be created and partitioned unless the related parent table exists. The Index creation is either part of the pre-transfer or will be initiated after all tables are filled with data. HiPAS decides automatically for the most suitable approach depending on the storage system and network bandwidth as described in Section 7-C.

B. *Adaptive Data Transfer*

The data transfer is based on two simple SQL statements:

- Insert into a table as selecting from a source table
- Querying remote tables through a database link

The combination of these statements makes it possible to fill local tables with remotely selected data. The resulting command is generated and parameterized at runtime:

```
sql_stmt := 'insert /*+ APPEND */ into "' || schema ||
'"."' || table_name || '" select * from "' || schema
|| '"."' || table_name || '"@' || db_link;
```

This statement is generated and executed by transfer jobs. The number of transfer jobs running in background is adapted continuously and depends on the resource utilization. As a pre-transfer stage, metadata of all objects stored in the source schema has been inserted into a table named "transfer_job_list". Tables to be transferred, exceeding a defined size, will be partitioned and, thus, transferred by multiple transfer jobs. In this case, the job type changes from "table" to "table_range" and row IDs mark the range's start and end (see Figure 5).

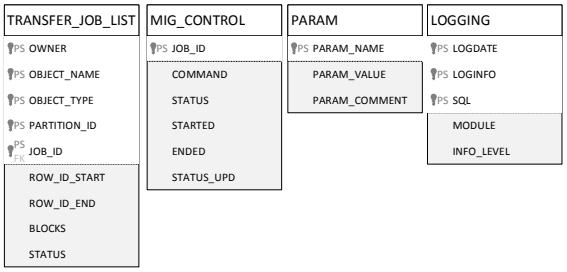

Figure 5. Metadata Entities for the Adaptive Data Transfer Phase.

Through partitioning, HiPAS can adapt more flexible to the current utilization, since the number of parallel jobs can be reduced or increased more frequently. HiPAS' table "mig_control" (see Figure 5) lists all background jobs transferring the objects stored in "transfer_job_list". In this respect the column "command" inside "mig_control" serves as an interface for controlling the transfer process, either autonomously by HiPAS or manually by the database administrator. When overwriting its content with keywords like "stop" or "continue", individual jobs will be stopped after finishing or continued, causing timestamps to be written into the column "status_upd" and if necessary into "ended". By this means, HiPAS is able to reduce or increase the number of parallel running transfer jobs transparently in respect of the optimizer's decision, which is described in Section 7. For the migration time, all constraints will be disabled temporary by HiPAS, enabling the table "transfer_job_list" to be ordered by blocks instead of considering key dependencies. Existing database triggers will also be disabled avoiding any unintended execution during the database migration.

C. *Post-Transfer*

After all source data has been transferred into the target schemas, the data has to be validated. Documenting data consistency and integrity is mission critical both for target database operation and for legal reasons. Only after verifying the equality of source and target data, the migration can be declared as successful, requiring HiPAS to not only compare source and target sizes, but also counting the rows of all tables. Finally, the disabled constraints and triggers will be enabled again.

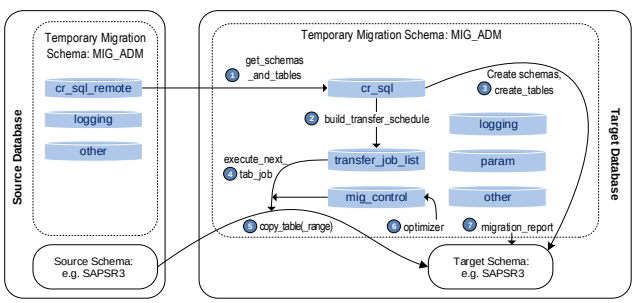

Figure 6. HiPAS Architecture.

VI. ENABLING PARALLELIZATION

In order to control the degree of HiPAS utilizing the available hardware resources the migration data transferred at the same time must be limitable. Restricting the number of parallel processed tables would be inappropriate since it required similar sized tables. Instead a defined number of blocks form a pack of data and a certain number of packs can be processed at the same time. That is, each pack has the same size and will be transferred by a single transfer job. Thus, adding or removing a transfer job burdens respectively disburdens the source and target system. HiPAS adapts to the underlying system resources by deciding autonomously how many transfer jobs are possible at any time.

To enable the amount of data to be partitioned into equal packs, a so called block split range defines their size. Since the tables on the target system are filled by generated "insert as select"-statements, its scope can be limited to a range between two row IDs, which represent the beginning and the end of each data pack. During the source schema analysis, these row IDs are identified by an analytical function. In this manner, large tables are partitioned into groups with row ID boundaries as Figure 7 shows exemplary.

```
GRP MIN_RID            MAX_RID
--- ------------------ ------------------
  2 AAAig0AAAAAABGAAAA AAAig0AAAAAABh/CcQ
  0 AAAig0AAAAAACAAAA AAAig0AAAAAAAl/CcQ
  1 AAAig0AAAAAAmAAAA AAAig0AAAAAABF/CcQ
```

Figure 7. Assigning Row IDs as Group Boundaries.

The identified IDs will be used during the transfer phase to limit the data of a single transfer job to the given block split range by adding a "where rowid between"-clause when selecting from the remote database:

```
insert into schema.table_name select * from
schema.table_name@db_link where rowid
between MIN_RID and MAX_RID;
```

Having partitioned the full amount of migration data into parts of a maximum defined size (block split range), HiPAS creates equally treatable transfer entities. These entities can be parallelized up to a degree defined by an adaptive transfer optimizer.

VII. ADAPTIVE CAPABILITIES

For parallelizing the data transfer during phase 2 of the migration process (see Section 5-B) with an optimal parallelization degree, we target an adaptive migration software. Adaptivity in general describes the capability of adjusting to an environment. In biology, the term is often used to describe physiological and behavioral changes of organisms in process of evolution. In informatics, the term is transferred to systems or components, which adapt to their available resources. However, here not to increase reproduction chances but often in order to achieve an optimal system performance. Adaption improves the resource efficiency and flexibility of software-intensive systems and means that a system adapts to changes of its environment, its requirements and its resources [11]. According to Martíin et.al. [12], adaption can also be seen as the first of three stages of the currently conceivable system complexity extent. Anticipation and rationality follow as further stages (see Figure 8).

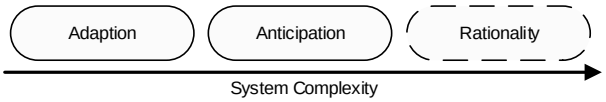

Figure 8. Levels of System Complexity (Adapted from [12]).

Thus, adaption describes the interaction of two elements: A control system and its environment. The goal is, to reach a defined state of the environment by means of actions initiated by the control system [13]. The control system then reacts on the self-precipitated changes of the environment with initiating new changes. It has been defined that an adaptive system is present, if the probability of a change of a system S triggered by an event E is higher than the probability of the system to change independently from the event:

$$P(S \rightarrow S'|E) > P(S \rightarrow S') \text{ [12]} \quad (4)$$

Furthermore, the condition has to apply, that the system reaches the desired state after a non-defined duration. This implies the convergence of the mentioned probabilities towards infinite:

$$\lim_{t \rightarrow \infty} P_t(S \rightarrow S'|E) = P_t(S \rightarrow S') \text{ [12]} \quad (5)$$

This law of adaption [12] requires the control system to know for each modification of its environment a sufficiently granulated attribute, which contributes to the desired state´s achievement allowing the adaption to end. For the first stage of complexity, the direction and the extent of modifications are built upon each other, thus, enabling the system to reach the desired state incrementally. If the modifications are not steps of a targeted adjustment process, but based on knowledge, predictions [14] or intuition, the process can be defined, in terms of system complexity, as anticipation. The third stage "rationality" implies intelligence; those systems are able to react to unpredictable changes of their environment and to balance contradictory objectives against each other [12]. This stage exceeds the objective of this paper and therefor was not scoped. Applying this differentiation on the design of an adaptive migration software, two approaches emerge for parallelizing the data transfer:

- A solely **adaptive** system, based on an incremental adjustment process, until changes do not evoke further improvements, thus, reaching the state of an optimal parallelization degree.
- An **anticipatory** system, which makes continuously new modification decisions independently of each other, based on knowledge about used and monitored resources.

 ISBN: 978-1-61208-378-0

These two approaches have been designed and implemented as described subsequently and evaluated as described in Section 8. Due to HiPAS' scalable architecture, the respective procedures could be implemented as plugins and additionally started for evaluation. Both plugins control the data transfer via values inside the table "mig_control" (see Section 5-B) serving as an interface.

A. Adaption

The solely adaptive approach will successively increase the parallelization degree and therefor the source and target systems utilization. After each enhancement its consequences on the system environment meaning the migration performance is measured in terms of inserted megabytes per time unit. The adaption can be started by running an additional procedure "calibrate", which invokes either the procedure "increase" or "decrease" for modifying the parallelization degree, starting from one transfer job per database link at the same time. The number of jobs to be added or deducted will be reduced after each time a change in direction was required, by this means the algorithm brings the number of parallel jobs closer to the optimum. After reaching a defined modification count (number of jobs to add or deduct) the algorithm assumes having approximated the optimal parallelization degree and the adaption ends, representing the finiteness requirement of adaptive systems.

The variable "diff_level" describes the current modification extent, meaning the number of jobs to start additionally or to stop after finishing. To reach a required level of flexibility for changing the number of jobs shortly, the size of a transfer job is limited to the introduced block_split_range. The following code example shows how the number of jobs is reduced by the value of the variable "diff_level":

```
update mig_control set command = 'STOP' where job =
'loop_while_jobs_todo' and command = 'continue' and
rownum <= diff_level; commit;
```

Since the tuples inside "mig_control" represent background jobs and each tuple has a row number, jobs can be stopped for each row number being smaller than "diff_level". The mentioned value "loop_while_jobs_todo" is the name of the procedure every background job runs for processing all defined transfer jobs listed inside the table "transfer_job_list". If a background job is marked with the command "STOP", it will be deleted after finishing the current transfer job and afterwards marked with the keyword "ended".

B. Anticipation

If the adaption is based on predictions, we call it anticipation as the next level of complexity [12]. In contrast to the solely adaptive approach, HiPAS now optimizes the parallelization degree continuously and based on a different algorithm. For mapping the described theoretical insights to our migration use case, we implemented an optimizer package, which predicts the optimal amount of parallel running jobs for the upcoming period. This decision is based on a combination of the following relevant system information, which is continuously monitored by the DBMS across all involved database instances:

- Concurrency events on target system
- Concurrency events on source system
- Average write time on target system
- Average read time on source system
- Average read time on target system
- Average write time on source system
- Redo log buffer size
- Available memory size

An important concurrency event, for instance, occurs when the high water mark of a segment needs to be increased, since new blocks are inserted into the same table by multiple and competing processes, this is known as high water mark enqueue contention [15]. The optimizer analyzes the above listed values and calculates a fail indicator as well as the number of additionally possible jobs according to the measured available resources like memory size and disk utilization. In contrary the fail indicator indicates possible bottlenecks and can prompt the optimizer to reduce the amount of currently running jobs. The introduced components form a feedback loop according to the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) loop reference model developed by IBM [16] as shown in Figure 9.

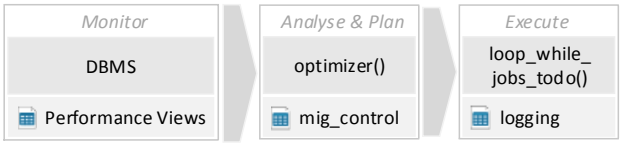

Figure 9. MAPE-K [16] Based Adaptive Feedback Loop.

Typical indicators for possibly arising bottlenecks are increasing concurrency events while the redo log buffer size decreases. If such a situation has been monitored, the optimizer will reduce the number of parallel jobs based on a high failure indicator. Whenever the optimizer acts, a log string is written to the logging table as in the following example:

```
"Prev Jobs: 40/ Jobs: 40 Max Jobs: 400 # Read Avg:
3.32(20-40) # Write Avg:  105.9(100-200) # R_Read Avg:
.12(20-40) # R_Write Avg:  .3(20-40) # R Fail Ind:  3
conc:3026(2607)              redo:5720763732(5776886904)
r_conc:5157(5069) # numjobs > 0 # Jobs being stopped:
0 # (Resource Overload) and numjobs > minjobs and
jobs_being_stopped = 0 # Running: 20/Stopping: 5 on
inst:1 # Running: 20/Stopping: 5 on inst:2"
```

In the above extracted example, 40 jobs are running in parallel. Due to increasing concurrency events, the optimizer detects a possible overload of the target system and decides to stop 5 running jobs on each instance. The jobs will terminate after they completed transferring their current objects. This is implemented by writing "stop" commands into the table "mig_control", which the procedure "loop_while_jobs_to_do()" will carry out (see Figure 9). The next log string will start with the information "Prev Jobs: 40/ Jobs: 30" accordingly. Additionally, not only the overall amount of jobs is measured, but also the memory each server process allocates. This value highly depends on the data

 ISBN: 978-1-61208-378-0

types of the currently transferred data. If too much memory is allocated, the number of jobs will be reduced as well. In order to avoid downward or upward spirals, e.g., due to the reducing redo log buffer size when stopping jobs, bottom lines and limits are defined. Hence the optimizer decides on the basis of a branched search for indicating relations between the monitored information. Surely, these are only indicators not to be seen as evidence, so the algorithm follows a heuristic approach. In contrary to the solely adaptive approach and to a statically parallelized transfer, the optimizer is able to dynamically react to unexpected events and predict a possibly optimum level of system utilization during the whole migration process. In the following Sections and for the evaluation, when mentioning the adaptive capabilities, we always refer to the anticipatory approach as it was performing more efficient during preliminary tests.

C. Time of Indexing

As previously termed as the "index problem", the right time for indexing the data depends on the combination of storage system performance and network bandwidth. If not manually parameterized inside the "param" table, HiPAS therefore decides by means of test tables filled with random data and having indexes on multiple columns, if it creates the indexes before or after data loading. For the two possibilities of index creation, the time for performing the respective steps is measured and compared to each other. After comparing the two measurements, HiPAS updates the parameter "index_while_transfer" inside the "param" table autonomously by inserting "true" or "false". This test can be performed during a common migration test run on the actual system environment and excluded for the productive migration reusing the "param" table.

D. Transfer Order and Instance Affinity

The table "transfer_job_list" contains all objects, which need to be transferred to the target. When selecting the next object for transfer, this table needs to be ordered by blocks since large objects are preferred by HiPAS. Furthermore, an instance prefers table partitions of tables, which already have been started to be transferred by this instance. Accordingly the next table or table range to be transferred is always selected as follow:

```
select * from transfer_job_list where status =
'PENDING' and object_type = 'TABLE' and (instance = 0
OR instance = sys_context('USERENV', 'INSTANCE'))
order by instance desc, blocks desc, partition_name;
```

If an instance starts transferring a table range of a large table, it marks all other table ranges of the same table by inserting the instance number into all tuples related to this table. By this means, instances reserve tables in order to avoid loading the same table into data buffers of other instances. For this reason, instances prefer tuples marked by themselves and tuples not reserved by any other instance, which has been implemented by means of the above displayed "where clause". In addition, the actual block split range, defining the limit for the size of all table ranges, is identified partly adaptively. For a given maximum block split range, HiPAS calculates the optimal block split range by counting tables and their sizes resulting in an optimal ratio of a ranges size and its total count.

VIII. EVALUATION

The migration method has been tested in several customer environments with differently powerful server, storage systems and networks. Following the design science approach, HiPAS has been improved in multiple iterations based on test results.

A. Experiment Setup

For this paper, we set up a test environment consisting of a source and target system installed on physically separated virtual machines, each having 4 CPUs and 16 GB of main memory. Both the source and target database are real application cluster (RAC) environments running Oracle Database 11g Enterprise Edition Release 11.2.0.3.0. On each side two instances are available connected to the other side through a 1 Gigabit Ethernet. The source system reads from solid state drives and the target system writes on common SATA disks. For evaluation, we performed multiple test runs belonging to the following three different main tests:

(1) Function test with a 300 GB schema (Test A)
(2) Performance test with a 16 GB schema (Test B.1)
(3) Performance test with a 32 GB schema (Test B.2)

To create the different database schemata, we implemented a software package, which generates database schemata filled with random data and including all special cases we could imagine HiPAS to encounter at productive customer environments. By means of this software, we created different sized test schemata inside the source database for test migrations. For the function test (Test A), the schema included characteristics like foreign key constraints, a variety of character, numeric and binary data types, reference partitioning, indexes, table clusters, views as well as different rights and roles. In this manner we were able to test the compatibility of HiPAS with different data types, objects and complex data structures. The used schema has an overall size of 300 GB, which was large enough to analyze HiPAS adaptive behavior during the migration run. To compare HiPAS migration performance with the current Oracle standard migration tool for exports and imports "Data Pump" [17], [10], we reduced the size for being able to perform multiple test runs and to average out performance values across all performed runs. These performance focused migration runs are referred to as test B. After each migration, we fully deleted the migrated schema and rebooted the whole server in order to have the same initial cache situation for all runs. The results of all tests are shown subsequently.

B. Results

In the following the results of the function test (A) and the performance tests (B) are presented.

1) Function Test (Test A): As described in Section 6-A, "Test A" aims at analyzing HiPAS adaptive behavior and compatibility. We implemented a package, which compares the created target schema with the original source schema by

 ISBN: 978-1-61208-378-0

counting rows and columns. We verified that all data objects were created inside the target schema successfully. The optimizer, providing the adaptive capabilities of HiPAS, writes log information whenever an adaption is needed. An example of such a single log string has been introduced in Section 7-B. Analyzing all tuples, written into the logging table during a migration run, leads to the migration process shown in Figure 10. The transfer started at 12:08 pm and ended at 12:47 pm. HiPAS transferred the created test schema, filled with 300 GB of random data, starting with 20 background jobs running in parallel meaning 10 jobs per instance, since HiPAS identified two available instances on the target system for job distribution. After 39 minutes, the transfer ended with a current total number of 116 parallel running jobs. The "block split range" was 25120 blocks, so, with a configured data block size of 8 KB, each job transferred a maximum amount of approximately 200 MB.

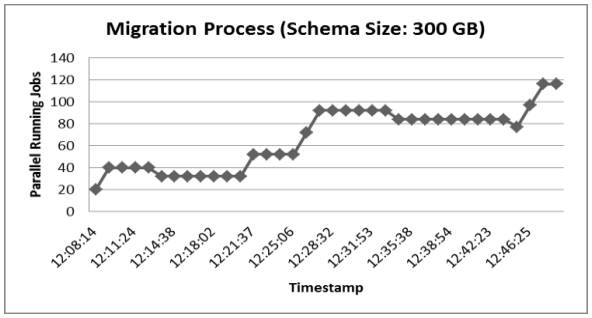

Figure 10. Adaptive Migration Process with HiPAS.

Tables, smaller than the split range, were not partitioned and transferred at the end of the migration, since large tables are preferred by the data selection algorithm. If a job transfers less data (small table), more parallel jobs are possible, so HiPAS raised the number of running jobs as the migration time goes by, which explains the slope of the graph shown in Figure 10.

2) Performance Test (Test B.1): For the first performance test, we created a schema of 16 GB including the mentioned data types in Section 8-A. The different test runs of test B.1 are described as follows:

(1) Migration by means of HiPAS adaptively and with enabled partitioning of large tables
(2) Migration by means of HiPAS with a static parallelization degree of 20 running jobs and enabled partitioning of large tables
(3) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and enabled partitioning of large tables
(4) Migration by means of HiPAS with a static parallelization degree of 10 running jobs and disabled partitioning of large tables
(5) Migration by means of HiPAS without parallelization (sequential) and with disabled partitioning of large tables
(6) Migration by means of Oracle Data Pump

We performed the described test runs three times in order to compensate statistical outliers, possibly caused by uninfluenceable events of the database management system or the operating system. This was necessary because the test runs had to be performed successively to provide the same environment for all tested methods. Afterwards, we calculated for each method the average total duration of the three runs. The final result is shown in Figure 11. The small test schema of 16 GB has been transferred by HiPAS averagely within 11 minutes, enabling adaptive capabilities (more precisely "anticipation") and partitioning of large tables. Transferring the same schema by means of the Oracle tool Data Pump, using the number of available CPUs as the "parallel" parameter [17], took averagely 53 minutes, which means a deceleration of approximately 382% compared to HiPAS. Comparing the different HiPAS migration runs with each other, it can be stated that parallelizing in general noticeably reduces the transfer duration, which is indicative for our assumption of utilizing the available resources more efficiently by parallelizing. Comparing test run 3 and 4 shows that partitioning large tables for the transfer barely improves the overall performance, since the partitioning feature was implemented to improve the flexibility of HiPAS when its optimizer needs to adapt quickly to changing resource availabilities. Thus, the adaptive migration run with enabled partitioning of large tables performed best in terms of downtime shortness.

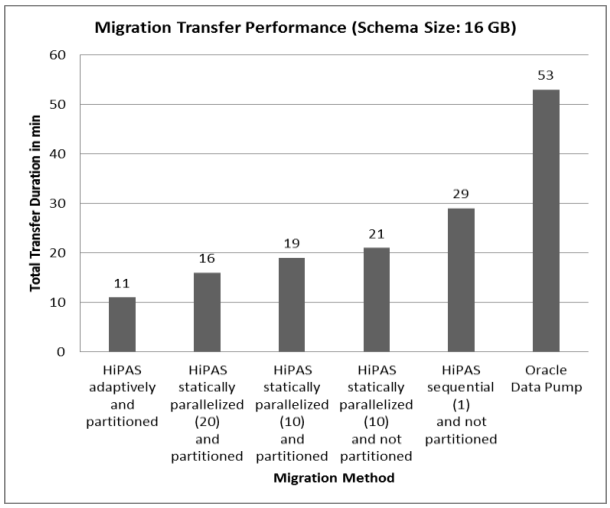

Figure 11. Transfer Performance for a 16GB Schema (B.1).

3) Performance Test (Test B.2): In addition to the 16 GB schema, we performed the same test runs with a schema size of 32 GB to evaluate how the adaptive capabilities work for a longer period of transfer time. The static parallelized runs have been performed as well and showed results proportional to test B.1, so we excluded them from Figure 12 on the next page.

 ISBN: 978-1-61208-378-0

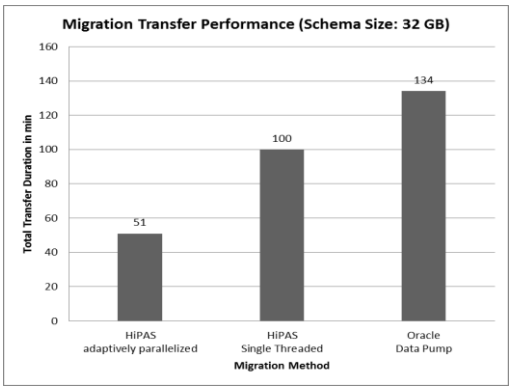

Figure 12. Transfer Performance for a 32GB Schema (B.2).

HiPAS, with enabled partitioning, adaptively transferred the schema within 51 minutes, compared to 2.23 hours needed by Data Pump, meaning this time HiPAS took 38% of Data Pump's transfer duration, whereas the single threaded configured HiPAS took about 75%. As a consequence, we assume, that non-adaptive sequential and Data Pump migrations leave useful resources idle or need to be tuned manually. In addition to the introduced test runs for evaluation within the scope of this paper, we performed several further tests in customer environments achieving considerable results, especially for schemata storing large objects. In terms of network bandwidth, we reached transfer rates of 120 MB/s for each database link created on a 1 Gigabit Ethernet.

IX. CONCLUSION AND FUTURE WORK

The target conflict of flexibility and performance, when choosing an offline database migration method, has been addressed by designing HiPAS. Through implementing an adaptive transfer algorithm, which continuously optimizes the source and target system utilization, significant performance gains have been achieved comparing the test results to non-adaptive migration methods. The paradigm of saving all migration metadata inside the database allows a clear and highly reliable architecture and appeared to support an efficient interaction of all HiPAS migration components and the actual migration data. We state, that implementing anticipatory capabilities into migration software significantly improve the performance of migrations invoked on database layer. Anticipation is more suitable than sole adaption, since database systems provide varied performance indicators, which need to be monitored during the whole progress.

Statically parallelized test runs did not adapt to changing utilization requirements, thus, performed less efficiently. The implementation as a stored object leads to the disadvantage of having to develop separate implementations for different database systems. As HiPAS currently has been implemented only for Oracle, we plan to build and evaluate further versions supporting different types of source and target systems. We encourage interested researchers to get in touch with us and share experiences in interconnecting adaptive components and databases.

ACKNOWLEDGMENT

We strongly like to thank all members of the Pasolfora Performance Research and Innovation Group (PPRG) for the support and possibility of performing the countless number of demo migrations during the development and evaluation of HiPAS. Furthermore, we thank Prof. Dr. Michael Höding of the Brandenburg University of Applied Sciences for giving scientific relevant input when mapping adaptive insights to the requirements of offline database migrations.

REFERENCES

[1] H.M. Sneed, E. Wolf, and H. Heilmann. Software Migration in Praxis. Dpunkt, 2010.

[2] M. Brodie and M. Stonebraker, Migrating Legacy Systems Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann, 1995.

[3] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy Information Systems: Issues and Directions". http://csis.pace.edu/~marchese/CS775/Proj1/legacyinfosys_directions.pdf, IEEE, 1999, p. 107.

[4] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research". MIS Quarterly Vol. 28 No.1., 2004, p. 80.

[5] M. Eastwood, J. Scaramella, K. Stolarski, and M. Shirer, Worldwide Server Market Revenues Decline -6.2% in Second Quarter as Market Demand Remains Weak, According to IDC. http://www.idc.com/getdoc.jsp?containerId=prUS24285213, 2013 .

[6] J. D. C. Little, "A Proof for the Queuing Formula: L= λ W, In: Operations Research", Cleveland, 1961.

[7] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, Fundamental Laws. http://homes.cs.washington.edu/~lazowska/qsp/Images/Chap_03.pdf, p. 42.

[8] G. Somasundarum and A. Shrivastava, Information Storage and Management - Storing, Managing, and Protecting Digital Information. EMC Education Services, Wiley Publishing Inc. Inianapolis 2009, p. 35.

[9] R.M. Karp, Reducibility Among Combinatorial Problems. In: Miller, R. E. and Thatcher, J. W. Complexity of Computer Computations. Plenum Press, New York 1972, 93.

[10] Oracle. Oracle Database Utilities 11g Release 2. 2014, p. 1

[11] Fraunhofer. Adaptive Systems. Fraunhofer Institute for Embedded Systems and Communication Technologies, http://www.esk.fraunhofer.de/de/kompetenzen/adaptive_syste me.html.

[12] J. A. Martin Hernandez, J. de Lope and D. Maravall, "Adaptation, anticipation and rationality in natural and artificial systems: computational paradigms mimicking nature.", Natural Computing, Volume 8, Issue 4, Springer Netherlands, 2009, pp. 758-765.

[13] N. Wiener, Kybernetik. Econ-Verlag, Düsseldorf 1963.

[14] R. Rosen, Anticipatory systems. Pergamon Press, Oxford 1985.

[15] Oracle. Enqueue: HW, Segment High Water Mark - contention. http://docs.oracle.com/cd/B16240_01/doc/doc.102/e16282/oracle_database_help/oracle_database_wait_bottlenecks_enqueue_hw_pct.html, 2009.

[16] IBM. An architectural blueprint for autonomic computing. Tech. rep., IBM. 2003.

[17] Oracle. Data Pump in Oracle Database 11g Release 2: Foundation for Ultra High-Speed Data Movement Utilities, p.2-27.

 ISBN: 978-1-61208-378-0

Evaluation of Software-Based Fault-Tolerant Techniques on Embedded OS's Components

Hosein Mohammadi Makrani[1], Amir Mahdi Hosseini Monazzah[2], Hamed Farbeh[3], and Seyed Ghassem Miremadi[4]
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran 1155-9517
Email: [1]makrani@ce.sharif.ir, [2]ahosseini@ce.sharif.edu, [3]farbeh@mehr.sharif.edu, and [4]miremadi@sharif.edu

***Abstract*—Software-based fault-tolerant techniques at the operating system level are an effective way to enhance the reliability of safety-critical embedded applications. This paper provides an analysis and comparison of five well-known recovery techniques, i.e., micro rebooting, recovery block, N-Version Programming (NVP), micro extension, and transactional extension for an embedded operating system's components, from performance point of view. These techniques are applied without any modification on the main architecture of the operating system. The techniques are implemented on a virtual ARM Integrator board which is emulated by the QEMU software (2.0.0) under the control of Embedded Linux operating system (3.9.0). The totals of 5000 software errors are ignited using a simulation environment. The results show that the recovery time overhead varies between 0.17% and 0.67%, and the performance overhead varies between 5.81% and 218.65% depending on the techniques.**

Keywords-embedded operating system; fault tolerant; recovery; performance.

I. INTRODUCTION

Nowadays, the embedded systems are employed as crucial control components in safety-critical and real-time areas such as medical devices, automobile, and aviation. To maintain the dependability of such applications, several fault tolerance techniques have been proposed in the recent decades.

In the recent years, the improvements in the performance of hardware devices have led to excessive attentions to software fault tolerance techniques. The software fault tolerance techniques can be implemented at the application code or operating system of an embedded system. Applying the fault tolerance techniques in an operating system allow the designers to develop their application without worrying about the dependability of the whole system. Hence, operating system approaches are more frequently used in embedded systems. However, the implementation of fault tolerance techniques at the operating system level may have side effects such as the impact on real-time behavior of the embedded operating system or resource restriction. Therefore, many constraints (especially form performance point of view) should be considered in selecting a recovery technique.

An operating system may crash during several error conditions including: software corruption, hardware malfunction, memory access violation, and executing illegal instructions. Most operating systems immediately stop their operations as soon as they encounter crucial errors in their hardware or software. Kernel panic in UNIX systems is a good example for such behaviors in operating systems.

Among the vulnerable parts of operating systems, *extensions* (which become widespread in commodity operating systems such as Linux) play an important role in the reliability of operating systems. Extensions are optional components which are presented in the kernel address space namely device drivers, network protocols, and file systems. Kernel may include different extensions, and failure in each extension may propagate to the other ones; hence, the dependability of kernel extension is highly important. Extensions cover up to 70% of the operating system source codes, and their error rate is calculated as 3x to 7x more than other source codes in operating systems [1].

Considering the above discussion, the goal of Fault tolerance techniques which are presented in this study is to recover from the transient errors which take place inside the embedded operating system extensions. The common characteristic of these methods is that they do not impose any modification on the base architecture of operating systems. Investigated recovery techniques are *micro rebooting*, *recovery block*, *N-Version Programming (NVP)*, *micro extension*, and *transactional extension*.

The contribution of this study is the evaluation of performance characteristics of well-known recovery methods on the same platform and operating system. The experimental results in this study will provide significant vision for the embedded system designer in using these recovery techniques. For each technique, the recovery time, the CPU utilization, the response time, and the performance penalty are compared with other techniques as well as the baseline operating system.

In this study, from the software point of view, *Embedded Linux* is selected as a target embedded operating system. From the hardware aspect, the modified operating system is executed on an *ARM Cortex A9* CPU, which emulated by *QEMU* [2]. It is noteworthy that the investigated techniques are generic and not architecture specific; thus the results can be regenerated by any other configuration.

To investigate the characteristics of each technique, the totals of 5000 software errors are ignited. The simulation results reveal that the recovery time overhead varies between 0.17% and 0.67%, and the performance overhead varies between 5.81% and 218.65% depending on the techniques.

The remainder of this paper is organized as follows: Section II describes error signaling and the component

 ISBN: 978-1-61208-378-0

isolation support for error confinement. Section III provides technical overview of the investigated techniques. The experimental setup is presented in Section IV and the results are presented in Section V. Finally, Section VI concludes the paper.

II. Error Detection, Containment and Signaling

In this section, error detection, component isolation, and error signaling support for error confinement will be described.

A. Error Detection

An error can be detected by different mechanisms in an embedded operating system. Virtual memory protection, processor exceptions, code checksums, and watchdog timers are some of the well-well known detection methods. To improve the reliability of an embedded operating system, besides the error detection methods, error containment methods should be considered as well. Employing error containment methods leads to the isolation of the erroneous part(s) of an embedded system from the other parts. After detection and containment of an error, as a final step, recovery technique can be applied on the affected component if the error is limited to its inside. The techniques which are under evaluation in this study are placed in the final step.

B. Error Containment

In the following paragraphs, the various isolation mechanisms are discussed in detail.

1) Isolating extensions by code:

A worthy project to isolate component in Linux is Nook [3] [4]. The Nooks isolation mechanisms avoid errors that occur in the extensions in order to affect the kernel. Each kernel extension in Nooks runs in "*light weight kernel protection domain*", which is considered for each kernel extension. Isolation mechanism can provide two main features for a system. The first one is to protect the domain from any manipulation. The other feature is "inter-domain" control transfer.

2) Isolating extensions by virtual machines

"*Virtual Machine*" is another method which isolates the extensions from the rest of a system [5]. In this method, when an extension is called, the unmodified version of that extension is run on its original operating system by a virtual machine. This mechanism allows wide reuse of exiting extensions, without considering the operating system. By running each extension in a virtual machine environment, this method isolates faults produced by faulty extensions. In addition, [6] and [7] utilized virtualization to confine extension in its virtual machine.

3) Isolating extensions by moving them to User-Space

The method introduced in [8] proposes to run extensions as unprivileged user mode. The results of this study reveal that extensions can be isolated without considerable performance degradation.

Besides the above methods, the Micro-Driver introduced new architecture which maintains critical time consuming codes in the Linux kernel and moves the reminder of the extension code to user-mode process [9]. Furthermore, in [10], user-level driver is implemented for Windows NT.

4) Compiler-level extension isolation

The Open Kernel Environment (OKE) project supports fully optimized code to be loaded in the kernel [11]. The OKE enables the restriction modification on the code executing in the Linux kernel. The Decaf Driver is another approach to develop drivers by modern languages such as Java [12]. In Decaf Driver, Linux extensions are converted to Java language and then executed in user mode.

5) Isolating extensions by changing architecture design

In a number of operating systems, a microkernel is implemented instead of using a monolithic kernel. Microkernel only provides simple kernel services. Other operating system functionality is transferred to the user space and does not execute at the privileged level. These architectures intrinsically increase the reliability of the system since each module can be individually controlled. MINIX3 [13], Mach 3.0 [14], Choices [15] and L4 [16] are some of the operating systems that benefit from microkernel architecture.

C. Error Signaling

Exception handling is usually employed to signal errors in user code. In the Linux kernel, the use of exception handling has been explored in [17]. Hence, system designers can write exception handlers to manage errors such as null pointers and invalid op-codes execution in the operating system. This allows designers to develop a flexible and robust technique to handle errors. Generic handlers only print out an error message and stop the operation of the system; however, local exception handlers generate a desirable response and try to recover from failures.

III. Recovery Techniques

The techniques which are introduced in this study can be implemented simply through software approaches on the operating system components. All these techniques are dealing with transient failures. In the following subsection, the architecture of these techniques and how they are implemented in our evaluation will be explored.

A. Micro Rebooting

Considering the terminology of micro rebooting, re-execution of the specific part(s) of an application (not the whole application) is called micro-rebooting. As expected, this technique uses time redundancy to recover errors. Micro rebooting can be applied in both application programs and/or operating system. For the first time, micro-rebooting was employed to recover faulty application components in [18]. The evaluation presented in [18] shows that employing micro-reboot increases the availability by reducing recovery time. This technique also can be used at operating system to recover faulty components [4]. In [15], it is shown that performing micro rebooting on faulty extensions is a simple and effective technique to enhance dependability of operating system.

In our implementation, micro-rebooting mechanism has two parts. The goal of first part is to bring the system and its extensions back into clean state. In this part, we insure that

 ISBN: 978-1-61208-378-0

resources are not taken after they released. In the second part, recovery mechanism runs user-mode recovery agent, which can set recovery policies for extensions before reloading them. Those policies can be written by users in the configuration files. The main task of recovery is to unload extension and load it again. When an error is detected by the detection mechanism, it signals to recovery agent and it runs recovery routine. After reloading the faulty extension by the agent, it signals the application to send its request again.

B. Transactional extension

Transactional extension is another approach to recover systems by using time redundancy. In database expressions, a transaction is a set of operations, which donate a unit of consistency and recovery. Features provided by transactions are isolation, failure atomicity and recoverability [19]. Transactional extension performs transactional operations with four features "ACID". These features are atomicity, consistency, isolation and durability [20]. Durability can frequently be ignored to simplify implementation.

In MARS project, a transactional model was exploited to define the activities of a real-time system [21]. The VINO kernel also used this technique to protect the kernel against misbehaved kernel extensions [22]. In addition, transactional component and micro rebooting are both used in the Choices. The Quicksilver distributed system is another project which exploits transactions [20].

In our transactional extension, before performing any operation, the state of the extension has to be saved. If an error occurs during the transactional operation, the state can be rolled back and the transaction will be aborted. Subsequently, the operation is re-executed. Applying transactional model on extensions leads to performance and space overheads. The need to save extension states before the operation commitment causes space overhead. Moreover, the time overhead is due to perform extra operations. Therefore, the overhead of this technique depends on the granularity of the operations.

There is a main difference between micro rebooting and transactional extension. The micro rebooting reloads extension and re-initializes its internal state. However, the transactional component only rolls back current transaction. Each of these techniques can be used depending on the extension. In general, if an extension has a large volume of data and many internal states, it is more efficient to use transactional techniques since occurring an error may harm the amount of data in micro rebooting; moreover the recovery process impose noticeable overhead. If data loss is not highly important, micro rebooting is a suitable candidate for recovery technique in terms of reducing overhead.

C. Recovery Block

The main recovery block structure is diverse software fault tolerance technique, which is categorized as dynamic techniques. The hardware fault tolerant technique related to the recovery block is stand-by sparing. This technique uses backward strategy to achieve fault tolerance.

In general, recovery block consists of two variants and one acceptance test. The first variant is called primary alternate or primary try block. Another variant called secondary alternates. These blocks are located in the series. In addition, real-time implementation of recovery block includes a software watchdog timer.

In the implementation of recovery block in this paper, the following extension and procedure are implemented:

- Primary extension, Secondary extension (it is equal to the primary), Manager Procedure, Save procedure, Restore procedure, Acceptance test procedure, and Send result procedure.

Our acceptance test is implemented as an application-dependent error detection mechanism such as reasonable check. The acceptance test is unique for two extensions and it includes no fault tolerance approach as it should be simple and quick. Watchdog timer procedure is used to detect irregular behavior such as infinite loop. The manager takes request from application program and saves state and request. Then, it sends request to primary extension. Simultaneously, it also starts a timer. If the response is not returned from primary extension, manager waits until timer trigs an exception. If the deadline is missed, manager unloads the primary extension, restores the state and issue a request. If the deadline is not missed, the manager sends request to acceptance test. According to the result of acceptance test, the manager decides to send the result to application or sends it to secondary extension (in the case of receiving error signals from acceptance test unit). If none of the extensions give correct response, the manager has to send error code to the application program.

D. N-Version Programming

The NVP is one of the well-known design diverse software fault tolerance techniques. The NVP is a static technique in comparison of recovery block. Since a task is executed by some programs, the result is selected among programs results via a majority vote.

In this paper, the NVP is implemented the same as Three Modular Redundancy (TMR). It means that we use three different versions of an extension. The difference of TMR with NVP is that, TMR cannot cover programming mistakes or bugs, because it uses three copies of a program, which are equal, but it can mask other types of errors as well as NVP. The benefit of NVP (with three version of a program) is that it can transparently recover or mask one error. If an error occurs and the detection mechanism detects it during execution of an extension, it sends a signal to voter, and then the voter omits the result of faulty extension. Like recovery block, NVP has a manager procedure which reloads faulty extension. The manager is responsible to take a request from applications and send the result back to them. Thus the application only interacts with one module.

E. Micro Extension

Micro extension is a combinational technique which includes: micro rebooting, transactional extension and user-level isolation mechanism.

The main goal of micro extension is to reduce kernel extension size, and increase reliability of operating system. This is done by moving some parts of extension to user space. In addition, its objective is to recover faulty extensions with the minimum overhead. To reach the goal, new approach is proposed which recovers only some parts of extension. This

 ISBN: 978-1-61208-378-0

technique neither is as fast as micro rebooting nor as slow as a technique which their whole extensions are fully in user space.

It was shown that 65% of extension operation can be moved to the user space [9]. Moving some parts of extension to user space is a kind of isolation mechanism. Micro extension also saves internal state of extension, before doing user space operations, just similar to transactions. At last, it should be noted that recovery of user level application which performs extension operations is similar to micro rebooting.

The difference between micro extensions recovery mechanism and micro rebooting is that micro rebooting unloads and reloads the whole extension without any restoring information; however, micro extension recovery mechanism only destroys and recreates the application which performs user-level operations on behalf of the extension. It should be noted that the extension is not changed any longer. Furthermore, this recovery is transparent from applications which have sent requests for extension. Additionally, this mechanism can restore internal state of extension (which is saved before invoking the extension operation from application) after application failure.

IV. EXPERIMENTAL SETUP

In this section, the experimental setup used in our implementations is presented.

A. Experimental Testbed

1) Operating system

Today, Linux is one of the most employed operating systems in embedded applications which deliver its service through GPL license. The ability to change the kernel in Linux-based operating systems made it possible for developers to customize the kernel by considering customer's demands. A noticeable portion of the introduced techniques is that they use a feature of Linux kernel called *Loadable Kernel Module (LKM)*. According to the above discussion, Embedded Linux is selected as a target embedded operating system in this study. The source code of Embedded Linux is available at [23].

2) Hardware configuration

QEMU as an open source machine emulator [2] is considered in the evaluation. In this paper, Cortex-A9 CPU (ARMv7) and Vexpress-a9 machine are chosen to emulate a system with 128MB of memory. This configuration provides a virtual ARM environment that runs Embedded Linux.

B. Error Activation

In this study, evaluation platform of recovery techniques requires error activation and detection units in order to signal error to error handler. Afterwards, the handler deploys appropriate technique to recover from the errors.

For the evaluation of recovery techniques, 1000 Software error activating experiments were performed for each of the five techniques. Fault model considered in this study to active errors are pointer dereferences, invalid arguments, and bad parameters which randomly injected in the extension. Table 1, depicts the faults model which were injected in the extension and their detection latency. Moreover, the response of system is reported when there is no fault tolerant technique.

TABLE I. FAULT TYPE

Fault location	Response of System	Detection Latency
Command's parameter of System Call	Error code	40960 clock cycle
Address's pointer of System Call	Error code	46336 clock cycle
Pointer of extension's internal function	Kernel panic	77632 clock cycle
Data structure of extension	Application termination and exception	177280 clock cycle
Computation of extension (to create infinite loop)	Kernel hang	200 ms
The application's data which is under control of extension	No signal	---

C. Test Methodology

The goal of this study is to perform a fair comparison among operating system-based fault tolerance techniques. To achieve this goal, a common workload should be considered for all the five techniques. Hence, an arbitrary extension with full controllability is written to explore the techniques considered in this study. Meanwhile, these five techniques can be applied to the real extensions.

The main task considered for the extension is arithmetical operation on matrixes. Three reasons can be enumerated in order to choose such task for extension. The first reason is that in several device drivers in order to increase computation speed, most computations are performed in the driver which is executed with high privilege and at the highest speed. For example, if a driver needs to calculate a parameter, there is no need to perform it at user level. Therefore, our extension can model these behaviors in a proper manner. The second reason is that the vulnerable part of the drivers is their computational part. The last reason is that blocks of data usually are transferred between an extension and an application as in network drivers. In this case, our extension can exchange large matrix with an application. Sorting algorithm is considered for extension task. Because NVP and recovery block need three and two different versions. The Bubble sort, the Insertion sort, and the Selection sort were selected. The primary task for micro rebooting, micro extension, and transactional extension is bubble sort.

To use this extension, a workload is needed to perform computation on matrix and work with them. Therefore, a data intensive application which can work with the extension and perform many computations on matrixes is constructed. This application is considered as the workload to evaluate the techniques. If the extension and the workload are changed, the comparative results cannot be changed a great deal. Hence, we try to report the comparative results (which expressed with percentage).

For measuring time, two mechanisms are used. "*Jiffies*" is used for measuring execution time, which is provided by Linux operating system. In the experiment, one "jiffies" is

 ISBN: 978-1-61208-378-0

equal to one millisecond. For precise measurement, the ARM cycle counter register (CCNT) which is provided by the processor is considered as well.

MR, TR, RB, NVP and ME are abbreviations which stand for micro rebooting, transactional extension, recovery block, N-version programming and micro extension, respectively. In addition WFT shows the average execution time of the application without considering any fault tolerant technique on the operating system.

In this paper, the performance overhead, the recovery overhead, the response time overhead, and the CPU utilization is reported. These parameters allow conducting deeper comparison from performance point of view.

V. Evaluation Results

In this section, the results of employing recovery techniques on the operating system are explored.

Before applying any fault tolerant techniques, the execution time of the application is measured. Moreover in this situation, the response time of the application request is measured. It should be note that these two values are the baseline values and any other result taken from the modified operating system will be compared with these values. It is evident that the QEMU has an impact on the performance, but it can be connived because its affect on all techniques is equal. Table 2 shows the application's execution time in different scenarios.

TABLE II. Application's Execution Time

		Techniques				
	WFT	***MR***	***TR***	***RB***	***NVP***	***ME***
Execution Time(ms)	7423	7855	8702	7897	23654	20004
Standard Deviation	17.0	21.4	41.9	22.1	18.8	21.9

(a) Average execution time without error activation

		Techniques				
	WFT	***MR***	***TR***	***RB***	***NVP***	***ME***
Execution Time(ms)	7423	7908	8734	7948	23695	20093
Standard Deviation	17.0	62.3	88.1	35.1	13.8	20.6

(b) Average execution time with error activation

As Figure 1 shows, the NVP has the maximum performance overhead, but it is not a bad feature. Since one request has to run on three extensions and after voting, the result is returned. Except the NVP, the result of Micro Extension seems incredible. This amount of overhead is related to operating system changing mode for each operation in one request. In fact, for each operation, the kernel extension runs an application in user space by means of API. This result reveals that it is inefficient to use Micro Extension technique when the extension is very computational. Therefore if the role of extension is changed, this can be expected that the performance overhead of Micro Extension technique will be changed as well. It means that the performance overhead is depended on the amount of extension's computational part in this technique. The minimum performance overhead belongs to Micro Rebooting. Transactional Extension has more overhead in contrast to Micro Rebooting and Recovery Block, because it has to save internal state before each transaction.

The response time overhead chart is similar to performance overhead one. It is clear that performance overhead of a workload which is created of several requests is directly related to response time of each request. Figure 2 shows response time overhead.

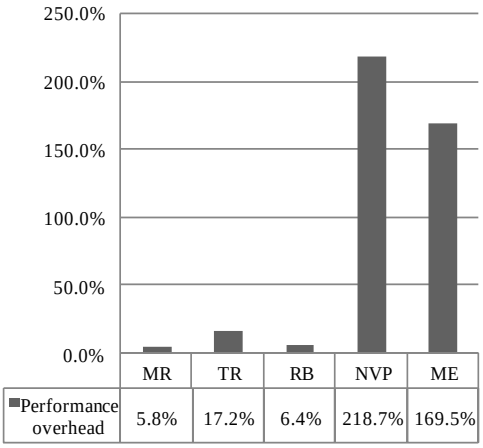

Fig. 1. Performance overhead

Figure 3 illustrates the recovery overhead of each technique. It is expected when a technique forces extra performance overhead, it provides better recovery overhead. The NVP has the best recovery overhead because nothing more is done by the technique when an error occurs. The NVP always masks one error. Since Micro Extension performs an operation in user mode, its recovery overhead is a little more than Transaction Extension. Regarding the CPU utilization, all techniques increase CPU utilization except Micro Extension, which is shown by Figure 4. It also decreases the CPU utilization. It happens because in the Micro Extension, some portion of time is devoted to context switch and transferring data between kernel and user-level part of extension.

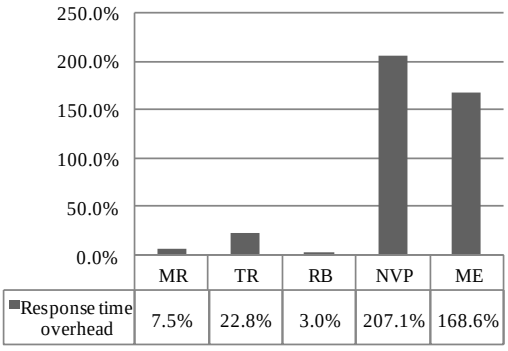

Fig. 2. Response time overhead

 ISBN: 978-1-61208-378-0

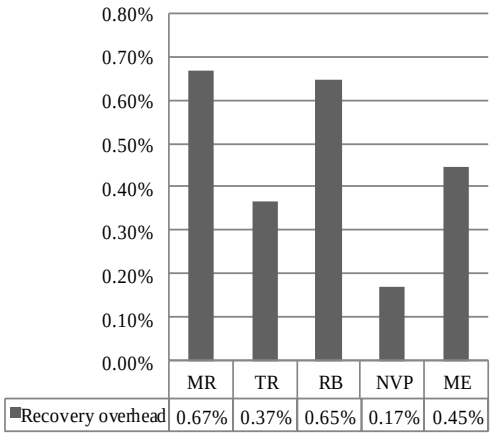

Fig. 3. Recovery overhead

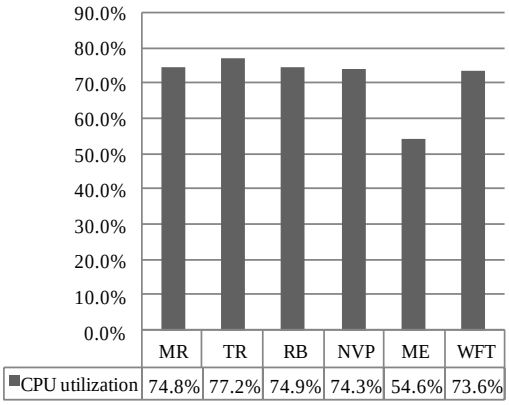

Fig. 4. CPU utilization

At last, the NVP has the best recovery time and the worst performance overhead. Furthermore, the Recovery block has the best response time and the Micro extension has the minimum CPU utilization.

VI. CONCLUSION

An analysis and comparison of five well-known recovery techniques, i.e., micro rebooting, recovery block, N-Version Programming (NVP), micro extension, and transactional extension for an embedded operating system are provided in this paper. These techniques are applied on the operating system extensions without any modification on the architecture of the operating system. This study investigates and compares the characteristic of those techniques on the same platform and operating system from performance point of view. This characteristic leads to an accurate and fair comparison among these methods.

The techniques are implemented on a virtual ARM machine which is emulated by the QEMU under the control of Embedded Linux operating system. The totals of 5000 experiments are made. The experiments results reveal that micro rebooting has the best performance overhead; otherwise, NVP has the worst performance overhead. In addition, the NVP has the best recovery overhead but micro rebooting has the worst one.

The simulation results are as follow: the recovery time overhead varies between 0.17% and 0.67%, and the performance penalty varies between 5.82% and 218.66% depending on the techniques. Additionally, extensions response time, in comparison with the base system, increases between 2.98% and 207.088%. Depending on the techniques, the CPU utilization is confined between 54.63% and 77.24%.

REFERENCES

[1] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, “An empirical study of operating system errors,” Proc. ACM Symp. Operating Systems, vol. 35, Dec. 2001, pp. 73- 88, doi:10.1145/502034.502042.

[2] F. Bellard, “QEMU, a fast and portable dynamic translator,” Proc. USENIX Annual Technical Conference, April 2005, pp. 41-46.

[3] M. Swift, M. Annamalai, B. Bershad, and H. Levy, “Recovering device drivers,” ACM Trans. on Computer Systems, vol. 24, Nov. 2006, pp. 333-360.

[4] M. Swift, B. Bershad, and H. Levy, “Improving the reliability of commodity operating systems,” Proc. ACM Symp. on Operating systems principles, vol. 35, no. 7, Dec. 2003, pp.207-222, doi:10.1145/945445.945466.

[5] J. LeVasseur, V. Uhlig, J. Stoess and S. Gotz, “Unmodified device driver reuse and improved system dependability via virtual machines,” Proc. Symp. on Operating System Design and Implementation, Dec. 2004, pp. 17-30.

[6] L. Tan, et al., “iKernel: Isolating buggy and malicious device drivers using hardware virtualization support,” Proc. IEEE Symp. on Dependable Autonomic and Secure Computing, Sept. 2007, pp. 134-144.

[7] T. Katori, L. Sun, D. K. Nilsson, and T. Nakajima, “Building a self-healing embedded system in a multi-OS environment,” Proc. ACM Symp. on Applied Computing, March 2009, pp. 293–298.

[8] B. Leslie, et al., “User-Level device drivers: achieved performance,” Journal of Computer Science and technology, vol. 20, no. 5, Sept. 2005, pp. 654-664.

[9] V. Ganapathy, M. J. Renzelmann, A. Balakrishnan, M. M. Swift, and Somesh Jha, “The design and implementation of microdrivers,” Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, March 2008, pp. 168-178.

[10] G. C. Hunt, “Creating user-mode device drivers with a proxy,” Proc. of the 1st USENIX Windows NT WS, 1997.

[11] H. Bos and B. Samwel, “Safe kernel programming in the OKE,” Proc. IEEE Conference on Open Architectures and Network Programming, June 2002, pp. 141-152.

[12] M. J. Renzelmann and M. M. Swift, “Decaf: moving device drivers to a modern language,” Proc. USENIX Annual Technical Conference, June 2009.

[13] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, “Construction of a highly dependable operating system,” Proc. IEEE European Dependable Computing Conference, October 2006, pp. 3-12.

[14] F. M. David, and R. H. Campbell, “Building a self-healing operating system”, in Proc. IEEE International Symp. on Dependable, Autonomic and Secure Computing, Sept. 2007, pp. 3-10.

[15] H. H¨artig, M. Hohmuth, J. Liedtke, S. Schonberg, and J. Wolter, “The performance of μ-kernel-based systems,” Proc. ACM Symp. on Operating System Principle October 1997, pp. 66-77.

[16] A. Forin, D. Golub, and B. Bershad, “An I/O system for Mach 3.0,” Proc. USENIX Mach Symp., 1991, pp. 163-176.

 ISBN: 978-1-61208-378-0

[17] H. I. Glyfason, and G. Hjalmtysson, “Exceptional kernel: using C++ Exceptions in the Linux Kernel,” October 2004. Available at: http://netlab.ru.is/exception/KernelExceptions.pdf

[18] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, “Microreboot – a technique for cheap recovery,” Symp. on Operating Systems Design and Implementation, vol. 4, Dec. 2004, pp. 31-44.

[19] J.N. Gray. “Notes on database operating systems,” In R. Bayer, R.M. Graham, and G. Seegmueller, editors, Operating Systems: An Advanced Course, Springer-Verlag, 1979, pp. 393-481.

[20] F. Schmuck, and J. Wylie, “Experience with transactions in QuickSilver,” Proc. ACM Symp. on Operating Systems Principles, October 1991, pp. 239-253.

[21] A. Damm, J. Reisinger, W. Schnakel, and H. Kopetz, “The real-time operating system of Mars,” Operating System Rev. July 1989, pp 141-157.

[22] M. I. Seltzer, Y. Endo, C. Small, and K. A. Smith, “Dealing with disaster: surviving misbehaved kernel extensions,” Proc. USENIX Symp. on Operating Systems Design and Implementation, October 1996, pp. 213-227.

[23] Embedded Linux source code, Available at: https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.9.tar.xz

 ISBN: 978-1-61208-378-0