SCIENCE • TECHNOLOGY • RESEARCH HIGHLIGHTS • VISIONS

Dissertation
50

Adaptive security in smart spaces

Antti Evesti

VTT

VTT SCIENCE 50

Adaptive security in smart spaces

Antti Evesti

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in lecture hall L10, at the University of Oulu, on the 31st of January 2014 at 12 noon.

ISBN 978-951-38-8113-9 (Soft back ed.)
ISBN 978-951-38-8114-6 (URL: http://www.vtt.fi/publications/index.jsp)

VTT Science 50

ISSN-L 2242-119X
ISSN 2242-119X (Print)
ISSN 2242-1203 (Online)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT
PL 1000 (Tekniikantie 4 A, Espoo)
02044 VTT
Puh. 020 722 111, faksi 020 722 7001

VTT
PB 1000 (Teknikvägen 4 A, Esbo)
FI-02044 VTT
Tfn. +358 20 722 111, telefax +358 20 722 7001

VTT Technical Research Centre of Finland
P.O. Box 1000 (Tekniikantie 4 A, Espoo)
FI-02044 VTT, Finland
Tel. +358 20 722 111, fax +358 20 722 7001

Kopijyvä Oy, Kuopio 2013

Adaptive security in smart spaces

Älytilojen mukautuva tietoturva. **Antti Evesti.** Espoo 2013. VTT Science 50. 71 p. + app. 119 p.

Abstract

Smart spaces – like smart homes, smart offices and smart cities – exploit various resources in order to offer enriched services and information for the end users. Achieving security in such a dynamic and heterogeneous environment with predefined and static security mechanisms is a challenging task. Hence, solutions for self-adaptive security are needed. Self-adaptive security is able to automatically select security mechanisms and their parameters at runtime in order to preserve the required security level in a changing environment.

The research problem of the dissertation is how to achieve security adaptation in a smart-space application. For this dissertation, architecture and knowledge base objectives were set. The objectives were satisfied with security-adaptation architecture that contains an adaptation loop and an ontology-based knowledge base for security. The adaptation loop conforms to the Monitor, Analyse, Plan, Execute and Knowledge (MAPE-K) model, which is a widely applied reference model in autonomic computing. The ontology-based knowledge base offers input knowledge for security adaptation. The research was carried using five cases, which iteratively developed the architecture and the knowledge base for security adaptation.

The contributions of the dissertation are: Firstly, reusable adaptation architecture for security is presented. The architecture strictly conforms to the MAPE-K reference model and defines all phases in it. Moreover, the architecture is the first that specifically separates security knowledge from the adaptation loop. Secondly, the architecture supports the utilisation of security measures to recognise an adaptation need. Security measures are presented by means of a three-level structure in order to achieve systematic monitoring. Due to the suggested architecture, it is possible to reuse and extend the defined security measures. Thirdly, this is the first time that an ontology has been applied for security adaptation. Hence, the Information Security Measuring Ontology (ISMO) acts as the knowledge base for the security adaptation. The ISMO is applicable at design-time and runtime alike. At design-time, the ISMO offers knowledge for the software architect, in order to design an application with security-adaptation features. In contrast, the application searches knowledge from the ISMO at runtime, in order to automatically perform the security adaptation. Utilising the ontology as a knowledge base ensures that the knowledge is presented in a reusable and extensible form. Moreover, the application does not need hard-coded adaptation knowledge.

Keywords Architecture, security measuring, ontology, knowledge base, self-adaptive

Älytilojen mukautuva tietoturva

Adaptive security in smart spaces. **Antti Evesti.** Espoo 2013. VTT Science 50. 71 s. + liitt. 119 s.

Tiivistelmä

Älytilat, kuten älykodit, älytoimistot ja älykaupungit, hyödyntävät monenlaisia resursseja tarjotakseen loppukäyttäjille parempia palveluita ja informaatiota. Muuttuvissa ja heterogeenisissa älytiloissa on haastavaa saavuttaa tietoturvaa ennalta määrätyillä ja muuttumattomilla tietoturvaratkaisuilla. Tämän vuoksi tarvitaan mukautuvaa tietoturvaa. Mukautuvassa tietoturvassa tietoturvamekanismit ja -parametrit valitaan automaattisesti suorituksen aikana, jotta vaadittu tietoturvataso saavutetaan myös muuttuvassa älytilassa.

Väitöskirjan tutkimusongelmana on, kuinka saavutetaan mukautuva tietoturva älytilan sovelluksessa. Tutkimusongelma on jaettu arkkitehtuuri- ja tietämyskantatavoitteiksi. Tavoitteet täytetään mukautuvan tietoturvan arkkitehtuurilla, joka sisältää mukauttamissilmukan ja ontologiapohjaisen tietämyskannan tietoturvalle. Mukauttamissilmukka noudattaa MAPE-K-mallia, Monitoroi (M), Analysoi (A), Suunnittele (P), Toteuta (E) ja Tietämys (K), joka on yleisesti käytetty referenssimalli autonomisissa ohjelmistoissa. Väitöskirjassa MAPE-K-mallin tietämysosa toteutetaan hyödyntäen ontologiapohjaista tietämyskantaa. Tutkimus on toteutettu käyttäen viittä tapaustutkimusta, joissa kehitetään mukautuvan tietoturvan arkkitehtuuria ja tietämyskantaa iteratiivisesti.

Väitöskirjan tulokset ovat: i) Työ esittää uudelleenkäytettävän mukauttamisarkkitehtuurin tietoturvalle noudattaen tarkasti MAPE-K-referenssimallia ja määritellen mallin kaikki vaiheet. Lisäksi arkkitehtuuri on ensimmäinen, joka täsmällisesti erottaa tietoturvatietämyksen mukauttamissilmukasta. ii) Arkkitehtuuri tukee mukauttamistarpeen havaitsemista tietoturvamittareiden avulla. Tietoturvamittarit esitetään kolmetasoisena rakenteena, joka mahdollistaa systemaattisen monitoroinnin. Lisäksi kehitetty arkkitehtuuri mahdollistaa tietoturvamittareiden uudelleenkäytön ja laajentamisen. iii) Työssä sovelletaan ensimmäistä kertaa ontologiaa tietoturvan mukauttamiseen. Tietoturvan mittausontologia (ISMO) toimii tietämyskantana mukautuvalle tietoturvalle. ISMO soveltuu sekä suunnitteluaikaiseen että suorituksenaikaiseen käyttöön. Suunnitteluaikana ISMO tarjoaa ohjelmistoarkkitehdille tietoa mukautuvan tietoturvan toteuttamiseksi osaksi sovellusta. Suorituksen aikana sovellus puolestaan etsii tietoa ISMO:sta suorittaakseen mukauttamisen. Ontologian käyttäminen tietämyskantana mahdollistaa tiedon uudelleenkäytön ja laajennettavuuden. Lisäksi sovelluksessa ei tarvita kovakoodattua tietämystä tietoturvan mukauttamiseen.

Avainsanat Architecture, security measuring, ontology, knowledge base, self-adaptive

Preface

The work presented in this dissertation was carried out in VTT Technical Research Centre of Finland. The work was performed in the following research projects: COSI (Co-development using inner & Open source in Software Intensive products), SVAMP (Software Variability Modelling Paradigm), SOFIA (Smart Objects For Intelligent Applications) and SASER-Siegfried (Safe and Secure European Routing) funded by VTT Technical Research Centre of Finland, Tekes (the Finnish Funding Agency for Technology and Innovation) and the European Commission. In addition, scholarships from the Oulu University Scholarship Foundation (Oulun yliopiston tukisäätiö) and the Finnish Foundation for Technology Promotion (Tekniikan edistämissäätiö) supported the dissertation work. I wish to thank the above institutions for making this research possible.

I'm grateful to Professor Jukka Riekki from the University of Oulu and Professor Eila Ovaska from VTT Technical Research Centre of Finland for supervising my dissertation. Moreover, I wish to thank Professor Ovaska for guidance and insights during the preparation of the original publications.

Professor Sam Malek and Professor Danny Weyns reviewed this dissertation. I appreciate their professional comments and suggestions, which have improved this work significantly.

I wish to thank the co-authors of the original publications Eila Ovaska, Reijo Savola, Susanna Pantsar-Syväniemi, Jarkko Kuusijärvi, Jani Suomalainen, Pekka Aho, Katja Henttonen and Marko Palviainen. Furthermore, I'm grateful to my colleagues who participated to the use-case implementations: Jarkko Kuusijärvi, Sakari Stenudd, Jussi Kiljander and Matti Eteläperä.

Finally, I wish to thank my wife Jenni for her patience and support during the years spent on this research.

Rovaniemi, 14th June 2013

Antti Evesti

Academic dissertation

Supervisor Jukka Riekki
Department of Computer Science and Engineering
University of Oulu

Reviewers Sam Malek
Department of Computer Science
George Mason University

Danny Weyns
Department of Computer Science
Linnaeus University

Opponent Kai Koskimies
Department of Software Systems
Tampere University of Technology

List of publications

This dissertation is based on the following original publications which are referred to in the text as PI–PVII. The publications are reproduced with kind permission from the publishers.

I Evesti A., Ovaska E., Savola R. From Security Modelling to Run-time Security Monitoring. In the Proceedings of the European Workshop on Security in Model Driven Architecture (SECMDA), Enschede, the Netherlands, 24 June 2009. Pp. 33–41.

II Evesti A., Pantsar-Syväniemi S. Towards Micro Architecture for Security Adaptation. In the Proceedings of the Fourth European Conference on Software Architecture (ECSA): Companion Volume, Copenhagen, Denmark, 23 August 2010. Pp. 181–188. DOI: 10.1145/1842752.1842790.

III Evesti A., Ovaska E. Ontology-based Security Adaptation at Run-time. In the Proceedings of the Fourth IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO), Budapest, Hungary, 27 September – 1 October 2010. Pp. 204–212. DOI: 10.1109/SASO.2010.11.

IV Ovaska E., Evesti A., Henttonen K., Palviainen M., Aho P. Knowledge based quality-driven architecture design and evaluation. In the Journal of Information and Software Technology, Elsevier, Vol. 52, No. 6, 2010. Pp. 577–601. DOI: 10.1016/j.infsof.2009.11.008.

V Evesti A., Savola R., Ovaska E., Kuusijärvi J. The Design, Instantiation and Usage of Information Security Measuring Ontology. In the Proceedings of the Second International Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS), Budapest, Hungary, 17–22 April 2011. Pp. 1–9.

VI Evesti A., Ovaska E. Design Time Reliability Predictions for Supporting Runtime Security Measuring and Adaptation. In the Proceedings of the Third International Conference on Emerging Network Intelligence (EMERGING), Lisbon, Portugal, 20–25 November 2011. Pp. 94–99.

VII Evesti A., Suomalainen J., Ovaska E. Architecture and Knowledge-Driven Self-Adaptive Security in Smart Space. In: Computers, MDPI, Vol. 2, No. 1, 2013. Pp. 34–66. DOI: 10.3390/computers2010034.

Contents

List of symbols

ASM	Adaptive Security Manager, a component utilised in the GEMOM adaptation approach
BM	Base Measure, the smallest raw measure, which is not dependent on other measures
CC	Common Criteria, ISO/IEC 15408
C1–C5	Use-case identifiers from Use-case 1 to Use-case 5 respectively
CO4SS	Context Ontology for Smart Spaces, context ontology that is utilised as a source for context information in this dissertation
COSI	Co-development using inner & Open source in Software Intensive products, one of the projects where this research work is performed
DB	Database, a storage utilised in the adaptation approach to offer input knowledge
ECA	Event-Condition-Action, one means to perform adaptation
ESCA	Event-State-Condition-Action, c.f. ECA
GEMOM	Genetic Messaging-Oriented Secure Middleware, the name of the project where the security-adaptation approach with the same name is developed
ISMO	Information Security Measuring Ontology, ontology developed in this dissertation and utilised as the knowledge base for security adaptation
KP	Knowledge Processor, an agent acting in the Smart-M3 based smart space
MAPE-K	Monitor, Analyse, Plan, Execute, Knowledge, a commonly used reference model in the autonomic computing field
MAPE	Monitor, Analyse, Plan and Execute, an adaptation loop in the MAPE-K reference model
NIST	National Institute of Standards and Technology, a non-regulatory agency in the U.S.
OIS	Ontology of Information Security, a security ontology utilised in the ISMO

OWL	Web Ontology Language, language to describe ontologies in a machine-readable form
PI-PVII	Identifiers for the original publications I-VII
pof	Probability of failure, a reliability value produced by means of RAP tool
QoS	Quality of Service, a quality indicator utilised specifically in communication and also applied in adaptation approaches
QPE	Quality Profile Editor, a tool developed in the SVAMP project to present quality requirements in UML profiles
RAP	Reliability and Availability Prediction, a reliability prediction method to calculate pof values as early as the architecture design phase
RDF	Resource Description Framework, a language to describe information formed to subject-predicate-object triplets
RDF-S	RDF Schema, an extension to RDF
RIBS	RDF Information Base Solution, one implementation of a Smart-M3 concept
SIB	Semantic Information Broker, the backbone element of the Smart-M3 concept
SMEPP	Secure Middleware for Embedded Peer-to-Peer systems, the name of the project that offered an environment for Use-case C1 of this dissertation
SMO	Software Measurement Ontology, the measurement ontology utilised in the ISMO
SOFIA	Smart Objects For Intelligent Applications, one of the projects where this research work is performed
SS	Smart Space, in this dissertation smart space is defined as a digital entity that presents and offers information from the physical world
SSA	Smart Space Application, an application constructed from a set of software agents
SUM-SS	Seamless Usage of Multiple Smart Spaces, the name of a pilot implementation that is utilised in use-case C5 in this dissertation
SVAMP	Software Variability Modelling Paradigm, one of the projects where this research work is performed
TLS	Transport Layer Security, a protocol that supports communication security
UML	Unified Modelling Language, a modelling language from the Object Management Group

1. Introduction

1.1 Background and motivation

Ubiquitous computing – envisioned in 1991 by Mark Weiser [1] – and smart spaces create dynamism and heterogeneity from the application viewpoint. Firstly, everyday appliances are networked and communicate with each other in smart spaces, and simultaneously, devices and services can appear and disappear. Moreover, smart space users are able to create new usage scenarios and utilise appliances in novel ways. Although several topics related to smart spaces have been researched, many unsolved problems and challenges have been recognised. Conti et al. named autonomic behaviour as an important challenge of pervasive computing and smart spaces in [2]. This challenge is related to the capability of devices and applications to adapt their behaviour as a response to changes in their operation environment. Similarly, achieving security in smart spaces is challenging. An application faces various environments and situations during its lifetime, which require different security objectives. For example, in some situations integrity is an essential security objective but in other situations authentication has the first priority. In parallel, the criticality of the handled information varies between situations. For example, applications may use entertainment information or more critical control information. Hence, the required security level varies from one situation to another. These variations and the dynamism of the environment are challenging for software developers; they cannot anticipate all possible changes and situations at design-time. Consequently, a smart space application must be able to adapt security based on the changing situations.

The vision of autonomic computing was presented by Kephart et al. in 2003 [3]. The authors envisioned that autonomic systems will maintain and adjust their operation in changing circumstances and when encountering failures. In contrast, Salehie et al. define self-adaptive software as a closed-loop system with the feedback loop aiming to adjust the system during its operation [4]. The purpose of adaptation is to maintain and adjust the operation of the system. Drivers for autonomic computing and self-adaptive software are, for instance, the complexity and heterogeneity of software. In other words, the set up, running and updating of software are resource-consuming tasks even for professional users [3,5]. The terms autonomic computing, self-management and self-adaptive are often utilised

interchangeably [4]. In this dissertation, the short term ‘adaptive’ will be used to refer to self-adaptive, which is a part of autonomic computing.

This dissertation focuses on security adaptation. The purpose of security adaptation is to maintain and adjust security in varying situations. This can be achieved by monitoring the attributes and actions which affect the required and achieved security. When a mismatch between the required and achieved securities is recognised security mechanisms are modified.

ISO/IEC 9126 [6] defines security as follows: “The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.” It can be seen that the definition contains the following security objectives: authentication, authorisation, confidentiality, integrity and availability. Conversely, Avižienis et al. define security as the composition of confidentiality, integrity and availability [7] and Common Criteria (CC) utilises a similar composition of security objectives [8]. In this dissertation summary, the term “security” refers to the set of security objectives included in the ISO/IEC definition.

A need to protect some assets creates a demand for security. CC defines an asset as an entity that someone presumably places value upon [8]. Thus, an asset can be almost anything that needs protection. Depending on the asset, different security objectives are required. For instance, control information requires that commands are not modified, and thus, integrity is required. On the other hand, personal data requires that outsiders are not able to read it, which sets a requirement for confidentiality. Consequently, the appropriate security mechanisms are needed to satisfy these requirements. In the literature, security mechanisms are also called controls, countermeasures and safeguards, which is due to overlapping terminologies and standards in the security area. In other words, there is no universal agreement on the many terms used in the security field [9]. Security mechanisms are a means to achieve the particular security objective. The National Institute of Standards and Technology (NIST) Special Publication 800-30 [10] divides security mechanisms into technical and non-technical. Technical mechanisms are included into hardware or software. Conversely, management and operational mechanisms are non-technical mechanisms. Therefore, the self-adaptation capabilities of software are related to the technical mechanisms.

In static security solutions, security mechanisms are selected at software design-time. Decisions are based on the assumptions from the forthcoming execution environment. However, it is difficult – or even impossible – to make these assumptions for software to be executed in heterogeneous and dynamic environments. Instead, the execution environment, the usage of software and the criticality of operations can only be recognised at runtime. Thus, the software must be able to adapt its security at runtime when information is available, which in turn makes it possible to achieve the required security even in heterogeneous and dynamic environments.

However, achieving software with security adaptation capabilities requires that several issues must be considered at design-time. Firstly, the means of how to monitor the environment and the software itself have to be designed and imple-

mented. Secondly, a meaningful interpretation of the monitoring results has to be made. Thirdly, the software has to contain variation points where different security mechanisms or parameters can be set at runtime. Consequently, a common and reusable architecture for security adaptation is needed. The architecture has to support the above-mentioned monitoring, result interpretation and variation of security mechanisms. An architecture, which defines these essential components, their purposes and mutual behaviour, facilitates the development of software with security adaptation capabilities.

Nevertheless, defining architecture for adaptation is not enough to achieve software with adaptation capabilities. In addition, appropriate knowledge is required at design and runtime alike. A software architect has to know what security mechanism supports the particular objective and what sensors to use to monitor the fulfilment of the security objective. Furthermore, the software itself requires knowledge at runtime to decide how to interpret the monitored results. Hence, the architecture for adaptation has to define where the required knowledge is stored and how to utilise it at runtime. When the software has recognised an adaptation need it has to decide the means by which to adapt itself in order to achieve the required security level. Naturally, all knowledge required at runtime can be hard coded in the application logic. However, hard-coded knowledge complicates software and knowledge reuse and maintenance. Consequently, separating the knowledge from the application logic and adaptation loop supports extensibility and reusability. Extensibility is required to respond to new vulnerabilities and threats and to extend the adaptation capabilities with new monitoring and reaction techniques. Reusability ensures that the existing knowledge and adaptation components can be easily reused, which speeds up software development.

A security adaptation approach, supported by a common and reusable architecture and the appropriate knowledge, makes it possible to achieve adaptive security with reasonable dynamicity in heterogeneous environments. A literature study [11] lists the challenges and future research needs in the field of self-adaptive systems. The authors recognise that applying self-adaptation to manage other quality attributes like security, usability and accuracy is one possible topic for future research. Consequently, the existing security adaptation approaches do not offer a complete means to produce software with security-adaptation capabilities. On the one hand, a survey by Elkhodary et al. [12] reveals that existing security adaptation approaches are not generic; instead the approaches concentrate on specific security objectives. In addition, the authors recognise that realising reusability and maintainability in the existing approaches requires additional research. On the other hand, a survey from Yuan et al. [13] shows that most of the existing approaches concentrate on the monitoring part of the adaptation loop, instead of covering the whole adaptation loop. Furthermore, the authors note that the architecture viewpoint is not covered by the existing approaches at a reasonable level. Utilising the existing approaches is complicated because the adaptation loop is not defined as a whole. Furthermore, the shortcomings of the architecture damage reusability and extensibility possibilities. The existing approaches apply the Monitor, Analyse, Plan, Execute and Knowledge (MAPE-K) reference model [3] implicitly, and thus,

the adaptation phases are mixed, or mutual relations are not defined. As mentioned above, the existing approaches cover the monitoring part of the adaptation loop. However, the approaches do not utilise a uniform way to present the applied monitoring techniques. Consequently, comparing and evaluating monitoring techniques is difficult. Lastly, the existing approaches do not cover the knowledge viewpoint at a sufficient level, and thus software designers are enforced to hard code the required knowledge. The MAPE-K reference model contains knowledge element K. However, the reference model does not define the form or utilisation of knowledge. In the existing approaches, the knowledge part is not taken into account at all, such as in the approach in [14]. Alternatively, a database for knowledge is added but its content is not described (e.g. the security-adaptation approach in [15]). The lack of the knowledge part means that the knowledge usage in the adaptation phases is not defined. The existing security-adaptation approaches are described in more detail in Sub-section 2.2.1.

In summary, several problems can be recognised from the existing surveys and security adaptation approaches. i) Common and reusable architecture for the security adaptation is not presented. ii) Reusability and extensions are not supported, which complicates the utilisation of approaches in dynamic and heterogeneous smart spaces. iii) The presented architectures do not support the separation of the knowledge part. Ignoring the knowledge part causes the utilised knowledge to be presented in a hard-coded form, which ruins the reusability and extension possibilities. These shortages set a need for the research presented in this dissertation.

1.2 Research objectives and scope

Based on the motivation described above, the research problem of the dissertation is: **How to achieve security adaptation in a smart space application?** This problem is divided into two main objectives. Solving these objectives will offer an answer to the research problem of the dissertation.

The first objective is to define the architecture for security adaptation. This objective is called, in short, the architecture objective. In order to fulfil this objective the following requirements have to be met in the architecture: The architecture has to conform to the MAPE-K reference model. Hence, the architecture has to contain the whole adaptation loop from the observations to the execution of adaptation and knowledge has to be separated from the adaptation phases. Moreover, support for reusability and extensibility is required. Finally, the utilisation of security measuring has to be supported in order to recognise an adaptation need systematically.

The second objective is to develop a knowledge base for security adaptation. This objective is called, in short, the knowledge base objective. The objective is fulfilled when the following requirements are satisfied: First, the knowledge base has to be applicable at design-time and runtime alike. At design-time, the knowledge base facilitates the design of an application with security adaptation features. At runtime, the knowledge base supports the adaptation loop in performing adaptation. Thus, the knowledge has to be presented in a form that makes the

knowledge reachable for the software architect and for the adaptive application. Secondly, it is necessary to know what knowledge is required in the different phases of the adaptation loop. Lastly, the knowledge base has to support reusability and extensibility in order to facilitate its usage in the various situations appearing in future smart spaces.

The fulfilment of the research objectives forms a basis for answering the stated research problem. The architecture contribution is built on the MAPE-K reference model and the knowledge base contribution is built by means of ontologies. The novelty of contributions comes from the integration, which is the first approach that combines the MAPE-K reference model, security measuring and knowledge from ontologies to consistent security adaptation architecture. In other words, the proposed solution: i) presents adaptation architecture for security that strictly conforms to the MAPE-K reference model and defines all phases in it. Moreover, the architecture is the first that specifically separates security knowledge from the adaptation loop; ii) supports the utilisation of security measures to recognise an adaptation need. Security measures are presented by means of a three-level structure in order to achieve systematic monitoring. Furthermore, the three-level structure makes it possible to define generic and implementation-specific parts for the security level monitoring; iii) is the first time that an ontology has been applied for security adaptation. The ontology-based knowledge base describes both security and measuring knowledge, and in addition, knowledge requirements in different adaptation phases are defined; iv) is validated from design-time and runtime viewpoints.

Applying the architecture and knowledge base, the software architect is able to produce smart space applications with adaptation features that support security in heterogeneous and dynamic smart spaces. Furthermore, smart space applications utilise the content of the knowledge base at runtime, in order to monitor the relevant attributes and to make the right decision.

Security adaptation can be applied in various domains, and thus, scoping and setting operational conditions for the research is important. From the security point of view, the adaptation approach has to be generic. In other words, it has to be possible to adapt various security objectives. Moreover, the approach has to apply existing security mechanisms, instead of developing dedicated mechanisms for the adaptation purposes. The research is performed from the end-user point of view. Hence, the purpose is to achieve security adaptation in smart space applications (SSA) and especially in the application parts executed in an end user's mobile device. In the context of this dissertation, the smart space application is defined as an application that is executed in heterogeneous smart spaces – such as a smart home, smart city or smart office environment – and utilises devices and information from the environment in order to facilitate end-user tasks. Thus, the end-user gets suitable notifications and is able to employ the surrounding devices. In other words, smart spaces are the domain in which the results are intended to be used. However, the results can be applied in other domains as well, but it is noteworthy that all runtime use cases are performed in a smart space environment. Lastly, security objectives (i.e. confidentiality, integrity, availability etc.) define the

research scope during the knowledge base definition. In other words, the content of the knowledge base has to be defined separately for each security objective because the required knowledge varies from objective to objective. In this dissertation, the content for the knowledge base is specifically defined from the user-authentication point of view.

1.3 Research approach and history

The research for this dissertation was mainly performed in the Smart Objects For Intelligent Applications (SOFIA) project during 2009–2011. Moreover, Co-development using inner & Open source in Software Intensive products (COSI) and Software Variability Modelling Paradigm (SVAMP) projects contributed to this work. The dissertation research belongs to the design-science research [16]. Therefore, the constructive research approach [16] is utilised.

Some results for this research were already achieved in the COSI project, in 2005–2008. In the COSI project, the author's work contained development of the Reliability and Availability Prediction (RAP) method and tool. The developed method and tool are utilised in publications PIV and PVI. In parallel with the COSI project, quality ontologies and software quality variability were researched in the SVAMP project in 2006–2007. The author's work was to develop a tool for managing quality attribute variability. The tool collects quality requirements and retrieves quality knowledge from quality ontologies. The first version of reliability and security ontologies were created and utilised in the SVAMP project. The research from the SVAMP project contributed to the publication PIV. Other publications are based on the results of the SOFIA project. The author developed the means to achieve security in dynamic smart spaces by using security adaptation.

Both the adaptation architecture and the ontology-based knowledge base are developed iteratively. The adaptation architecture is researched in publications PI, PII, PIII, PV and PVII. Simultaneously, ontologies and knowledge issues are researched in all original publications. The contributions of these publications are described in more detail in the next sub-section. The constructed artefacts are validated by means of use cases and a pilot implementation, which are listed in **Table 1**. The table contains an identifier (ID) for each case, connections to the related research objectives and case description.

Table 1. Use cases.

ID	Contributes to	Case description
C1	Knowledge base	The Secure Middleware for Embedded Peer-to-Peer systems (SMEPP) case concentrates on the design-time viewpoint. This case validates how a software architect utilises quality ontologies as a knowledge source to define quality requirements for the software under development.
C2	Architecture Knowledge base	The Smart Greenhouse demonstration contains a miniature greenhouse, sensors and actuators. Two user groups – gardeners and customers – act in the greenhouse. The case validates the first version of the adaptation loop. Risk-based security measures for confidentiality and integrity are utilised for security monitoring, and the required knowledge is retrieved from the ontology.
C3	Architecture Knowledge base	This use case validates the Information Security Measuring Ontology (ISMO). The case occurs in a smart home environment. Firstly, the ISMO is exploited at design-time. A software architect selects security mechanisms to support security requirements. Moreover, the architect retrieves suitable security measures to trigger security adaptation. Secondly, the application retrieves knowledge from the ISMO at runtime; the application monitors the user's authentication level by means of security measures and interprets the measured results by means of analysis models retrieved from the ISMO.
C4	Knowledge base	The ISMO Extension case illustrates how the ISMO can be extended with new knowledge. A software architect implements an application with security adaptation features. At design-time, RAP method is utilised to predict probability of failure (pof) values for the designed security mechanisms. These pof values are offered for runtime security adaptation purposes by means of the ISMO. The case presents how new measures and instances for the implemented security mechanisms are added into the ISMO.
C5	Architecture Knowledge base	The Seamless Usage of Multiple Smart Spaces (SUM-SS) pilot validates the whole adaptation architecture and usage of the ISMO in a heterogeneous environment. The use case contains fours smart spaces: a smart personal space, a smart home, a smart office and a smart city. A home owner is able to control home devices with her mobile device either locally or remotely. Different actions set individual requirements for user authentication, and thus, adaptation is required.

The timeline in Figure 1 presents the history of this research. The timeline contains projects (green bars), use cases (blue bars) and publications (yellow octagons). Furthermore, some main topics researched during these years are mentioned on the top of the figure. In COSI and SVAMP projects the focus was on design-time and architecture issues. However, the SOFIA project drove the focus towards runtime and smart space viewpoints. Publications are numbered in a chronological order based on the publication date. Publications PIV and PVII are

journal articles, and thus, the time frame from an actual research topic to the final publication is longer.

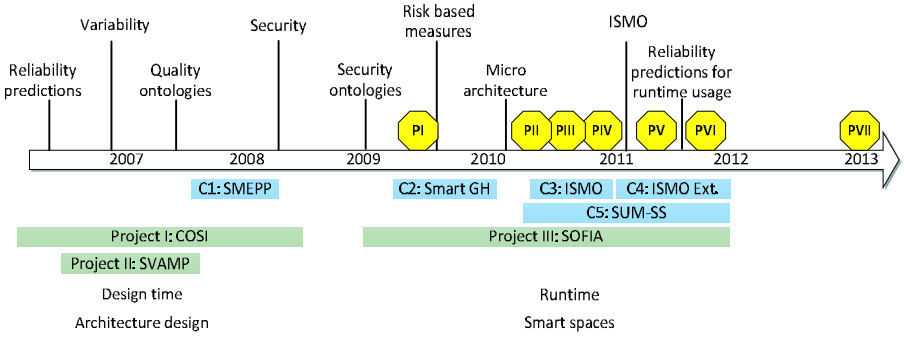


Figure 1. Research timeline.

1.4 Scientific contributions

The dissertation is composed of seven original research publications, which were published in between 2009–2013. Two publications were published in scientific journals and five in international conferences and workshops. The author of the dissertation was the first author in six publications and the second author in one publication. The contributions to each publication and the roles of writers are described below. In addition, author contributions are summarised in **Table 2**. Professor Eila Ovaska provided comments, insights and discussion for all of these publications. However, this is not repeated separately for each publication.

Publication I compares security ontologies from runtime-applicability and security-measuring viewpoints. The conclusion was that security ontologies for runtime usage exist, especially for service discovery and matchmaking purposes. However, there isn't any security ontology that describes security measuring in detail. Moreover, the publication presents the initial vision of the security adaptation concept. This concept is developed further in the next publications. The author is the main writer of the publication. Mr Reijo Savola commented on the publication and contributed to the security-measurement section.

Publication II suggests two important artefacts for the runtime security adaptation. Firstly, a taxonomy of context information for security is presented. The taxonomy contains concepts that affect the required and achieved security in smart spaces (e.g. smart space type, the utilised platform, the user's role and the role of exchanged data). Moreover, few initial mapping properties for the ISMO are introduced. The ISMO does not exist at the time of writing the publication, and thus, only the term 'security ontology' is used. The second presented artefact is the micro-architecture for security adaptation, which is an enhanced version of the initial vision presented in PI. The micro-architecture describes the phases required for security adaptation and information flows between these phases. Brief descrip-

tions of phases and information flows are also presented. The author is the first writer of the publication. Susanna Pantsar-Syväniemi commented on the publication and contributed to the context ontologies and context modelling parts. In addition, she was responsible for the context monitoring part of the micro-architecture. Pantsar-Syväniemi utilises these parts in her dissertation work [17].

Publication III suggests an ontology-based adaptation approach with risk-based measures. The block diagram of the adaptation is presented – showing the required steps before executing the adaptation. Adaptation is divided into two phases. The first is executed when the smart space application is started or joins the smart space – this phase is called the start-up phase adaptation. The second phase is called runtime phase adaptation, which is executed when security measuring reveals that the required security is not being achieved anymore. In this publication risk-based measures are used by measuring threat levels. Certain actions increase threat levels in the smart space and the used security mechanisms decrease these threats. C2 validates the adaptation approach, usage of the ontology and risk-based measures. The author is the first writer of the publication.

Publication IV suggests the quality-aware software architecting approach and the supporting tool chain. The approach contains three main steps: 1) modelling quality requirements, 2) modelling software architecture and transforming requirements to the models and 3) quality evaluations. Steps one and two are further divided into knowledge engineering and software engineering processes, whereas, the quality-evaluation step is divided into evaluations for evolution and execution qualities. Quality ontologies are one of the knowledge sources utilised in the approach. Consequently, in this publication the term 'knowledge base' does not only apply to ontologies. The work utilises earlier-developed quality ontologies for security [18] and reliability [19]. Both of these ontologies contain terminology that describes quality attributes, and in addition, measures for setting the required quality levels and measuring the fulfilment of quality requirements. Hence, the content of these quality ontologies and the ISMO follow the same structure, that is, the measuring and quality-attribute parts. From the scope of this dissertation, the publication contributes to usage of ontology-based knowledge base at design-time. In addition, C1 validates design-time usage of the ontology-based knowledge base. Professor Eila Ovaska is the first writer of the article and the author is the second writer. The author's contribution is related to quality ontologies, quality requirements modelling and quantitative quality evaluations for execution qualities. Furthermore, the author contributed to the related tool development and validation.

Publication V suggests the knowledge base called the ISMO and architecture elements for security monitoring. The publication describes the design and instantiation of the ISMO. Moreover, design-time and runtime usage of the ISMO is described. The ISMO is composed from two existing ontologies; the Software Measuring Ontology (SMO) [20] offers measuring-related terminology and the Ontology of Information Security (OIS) [21] provides security-related concepts. The ISMO draws mappings between these two ontologies on two levels. Firstly, concept-level mappings are defined. Thereafter, additional mappings are created by means of instantiated security measures. The ISMO is the first ontology that

combines security and measuring terminology on a detailed granularity level. Measures for password-based authentication are utilised as an example during ontology instantiation. The ISMO is a knowledge base for a software architect, in order to design an application with security-adaptation features. Moreover, the ISMO is a knowledge base for the smart space application, in order to perform runtime security adaptation. The ISMO is described by means of Web Ontology Language (OWL), and thus, applications are able to retrieve its content for adaptation purposes. The ISMO offers a generic and reusable manner to present diverse security measures. Furthermore, the ISMO ensures that security measures and terminology can be updated easily. The author is the first writer in this publication. Moreover, Reijo Savola and Jarkko Kuusijärvi have commented and contributed to the publication. Savola contributed to the security measurement issues and Kuusijärvi contributed to the validation use case.

Publication VI shows how design-time reliability predictions are brought for the runtime security-adaptation purposes. Firstly, the publication presents the required design steps to achieving security adaptation. After these steps, the RAP method is utilised to predict the forthcoming reliability from the software architecture. The RAP method produces probability of failure (pof) values for software components. These prediction results are stored by means of the ISMO, which ensures that the results can be utilised at runtime. However, this requires that the ISMO is extended. C4 validates the extension possibilities of the ISMO by defining extension needs and designing the required extensions. The author is the first writer in this publication.

Publication VII suggests a reusable security-adaptation approach, which can be used to adapt various security objectives. The presented approach is mapped to the MAPE adaptation loop. Hence, the approach defines the whole loop required in the security adaptation. The adaptation approach retrieves all the required knowledge from the ontologies, namely the ISMO and Context Ontology for Smart Spaces (CO4SS). Utilising the ontologies as the knowledge base ensures that the amount of hard-coded adaptation decisions can be minimised, and new knowledge can be added to ontologies without modifications to source code. Thus, smart space evolution does not cause changes to the smart space application's source code. Lastly, the publication presents the RDF Security Model, which provides an interoperable solution to control access to semantic information in smart spaces. Therefore, access control is handled in a dynamic way, which ensures that static and pre-defined access control lists are not needed in smart spaces. Instead, new devices are able to join the smart space without additional administrative efforts. The article is the joint result made together with Jani Suomalainen and Eila Ovaska. The author is the first writer in the article. The author has worked with two first-mentioned contributions (i.e. the adaptation approach and knowledge base). Simultaneously, the RDF Security Model was researched by Jani Suomalainen.

In **Table 2** the contributions of the publications are mapped to the State of the Art, Concept and Validation research phases. In addition, the table summarises the publications' contributions to these phases. In the validation column C1–C5 refers to the case IDs presented in **Table 1**.

Table 2. Contributions of the original publications.

No.	Research Phase		
	State of the Art	**Concept**	**Validation**
PI	Security ontologies.	The initial vision of the security-adaptation concept.	
PII	Context ontologies.	Micro-architecture for security adaptation and the taxonomy of context information for security.	
PIII	Few security adaptation approaches.	An ontology-based adaptation approach with risk-based measures.	C2 validates the adaptation approach, the usage of ontology and risk-based measures.
PIV	Quality-attribute ontologies for design-time usage.	Usage of an ontology-based knowledge base at design-time.	C1 validates the design-time usage of an ontology-based knowledge base.
PV	Software measurement terminology.	A knowledge base called the ISMO. Architecture elements for security monitoring.	C3 validates usage of the ISMO.
PVI		Design steps to achieve security adaptation.	C4 validates the extension possibilities of the ISMO.
PVII		A security-adaptation approach with the whole adaptation loop and utilisation of the ISMO.	C5 validates the adaptation approach and the ISMO as a whole.

1.5 Benchmark criteria

The security-adaptation approach presented in this dissertation is benchmarked with the related approaches in order to emphasise contributions and their validity. The attributes of the benchmarking criteria are described in **Table 3**. The benchmarking criteria are utilised in Sub-section 4.3, which presents a comparison to the related work.

Table 3. Benchmarking criteria.

Criterion	Description
Adaptation loop	What phases of the MAPE adaptation loop are covered?
Structure / Behaviour	Does the approach contain both architecture descriptions?
Supported security adaptation	What security objectives can be adapted? Alternatively, objectives are not restricted and the approach is generic.
Adaptation need recognition	How does the adaptation approach recognise the need for adaptation?
Extensibility	How can the approach be extended?
Reusability	How can the approach be reused?
Design / runtime	Does a support for design-time and runtime purposes exist?
Knowledge storage	Where is the required knowledge stored?
Knowledge form	What form is used to store knowledge?
Knowledge content	What knowledge is defined?
Knowledge usage	How is knowledge used?

1.6 Structure of the dissertation

This dissertation summary is divided to in five chapters. After the introduction, Chapter 2 presents the background and related work. Chapter 3 summarises the research contributions of the dissertation based on the original publications. Thereafter, Chapter 4 discusses the achievement of research objectives, constraints and future work. Finally, conclusions close the dissertation. The original publications are reprinted as appendices at the end of the dissertation.

2. Background

In this chapter, the main concepts are explained in the first sub-section. Thereafter, existing security-adaptation approaches and security ontologies are described in Sub-section 2.2.

2.1 Main concepts

2.1.1 Smart spaces

The desire to build smart spaces that offer enhanced services, which utilise information intelligently has existed for a long time [22]. The terms *smart space* and *smart environment* are widely used interchangeably. Cook et al. define a smart environment as one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment [23]. Conversely, Ovaska et al. define a smart environment as the composition of a smart space and a physical environment [24]. Simultaneously, the smart space is defined as a digital entity presenting and offering information from the physical environment in an interoperable and machine-readable form [24]. Consequently, the definition separates the terms smart space and smart environment. This dissertation contributes to the smart spaces, that is, the digital part, and thus, the latter definition is utilised.

Interoperability is the capability of a smart space to share and acquire information between smart space devices. The interoperability levels, defined in [24,25], describe the type of achieved interoperability. Conceptual interoperability is the highest interoperability level, which builds the smartness by utilising the data, context and actions from the smart space.

A smart space application is constructed from a set of software agents. Agents can be distributed to several smart space devices, and thus, the application deployment is distributed. In smart spaces, the smart space application offers enriched information and services for the end user. Context information and context changes have an important role related to the capability of the application to enrich the services and offered information. Chen et al. make the following definition: Context is a set of environmental states and settings that either determines an application's behaviour, or in which application the event occurs, and is interesting

to the user [26]. The context information makes it possible to select the most suitable information and services for the end user's purposes. Similarly, context information facilitates achieving security in the smart space. In other words, exploiting context information makes it possible to select a suitable security mechanism for the user's situation and action.

Raychoudhury et al. survey in [27] middleware solutions for pervasive computing, which are applicable solutions to offer infrastructure for smart spaces. However, in the SOFIA project – and also in this dissertation – *the Smart-M3* [28] concept is utilised as the smart space infrastructure. In the Smart-M3, a Semantic Information Broker (SIB) forms the backbone for the smart space. The smart space contains agents, namely Knowledge Processors (KPs), which share information with each other by means of the SIB. Hence, KPs do not communicate directly but insert, query and subscribe to information from the SIB. Thus, the Smart-M3 adopts the Blackboard architecture pattern [29]. The SIB and KPs present information in the semantic form by means of a Resource Description Framework (RDF) [30]. The first Smart-M3 implementation [31] was utilised in the greenhouse demonstration [32] (i.e. the case in C2 and PIII). The second Smart-M3 implementation is called the RDF Information Base Solution (RIBS) [33], which is designed for resource-restricted devices with Transport Layer Security (TLS) [34] support. The RIBS is utilised in cases C3 and C5.

2.1.2 Security

In this dissertation, the security definition is based on the ISO/IEC definition, and thus, security is thought to be a composition of the following security objectives: authentication, authorisation, confidentiality, integrity and availability [6].

Figure 2 presents security concepts and their relationships. In the scope of this dissertation, *asset* is data in a communication channel and data stored or processed in smart space devices. The content of the data is not restricted in this work; data can be, for instance, a message payload, log information, code, rules or a device identifier. Depending on the asset different *security objectives* are required due to potential *threats*, which in turn increase *risk*. A threat is defined as follows: a threat is potential for an accidental or intentional security violation [10,35]. Threats are enabled by *vulnerabilities*. Based on [10,36] a vulnerability can be defined as a property or weakness in a system or its environment that could cause a security failure. Lastly, *security mechanisms* are a means to protect assets and to support the particular security objective by mitigating risks.

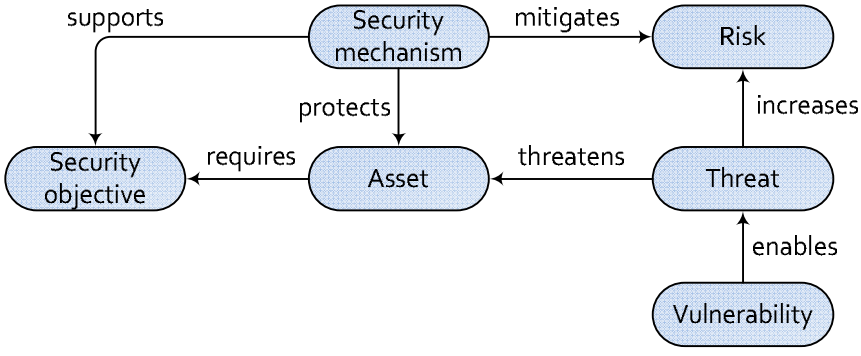


Figure 2. Security concepts.

Risk consists of the probability of threat realisation and the impact of the threat realisation (i.e. asset value) [10]. Consequently, risk management contains risk identification, risk assessment and risk mitigation steps to reduce the risk to an acceptable level [10]. Risk assessment requires quantitative values or categories in order to estimate threat probabilities and impacts to produce risk levels. Table 4 shows an example of the risk-level matrix, which contains fine-tuned risk levels from the original version presented in [10]. The risk-level matrix in Table 4 contains three rows for threat probabilities and three columns for impacts. Furthermore, there are three levels (low, medium and high) for probabilities, impacts and risk levels. For example, a threat with medium probability and high impact directs to the cell that states that the risk-level is high. However, the amount of levels and rating risks to levels is a subjective decision, and thus, these vary between references and the purpose of use – confer risk-level matrixes from [10,37]. Table 4 contains also continuous values for probability, impact and risk, which are utilised in use case C2.

Table 4. Risk-level matrix.

Threat probability	**Impact**		
	Low (10)	Medium (50)	High (100)
High (1.0)	Medium (1.0*10 = 10)	High (1.0*50 = 50)	High (1.0*100 = 100)
Medium (0.5)	Low (0.5*10 = 5)	Medium (0.5*50 = 25)	High (0.5*100 = 50)
Low (0.1)	Low (0.1*10 = 1)	Low (0.1*50 = 5)	Medium (0.1*100 = 10)

As described above, risk assessment utilises levels to define comparable categories for risks. Similarly, this dissertation uses a term *security level*, which is defined

for the dissertation as the security mechanism effectiveness to support the required security objective. The NIST report [38] defines the terms *security correctness* and *security effectiveness* as follows: The security correctness is the ability of the security mechanism to work precisely as specified, whereas, the security effectiveness is the security mechanism's strength to withstand attacks in carrying out their function. Consequently, the overall security level of the system has to take both correctness and effectiveness into account. Nevertheless, the effectiveness-based security level is utilised in this dissertation for the runtime security-adaptation purposes. The security correctness aspects are considered in PVI, that is, the reliability of the security mechanism. Moreover, this dissertation derives security objective specific levels (e.g. the authentication level) when emphasising the security level of the particular security objective.

2.1.3 Self-adaptive software

Self-adaptive software is able to modify itself based on changes in the execution environment, requirements, or in the software itself at runtime [39]. The MAPE-K model was presented simultaneously with the autonomic computing vision [3]. The MAPE-K model is applied as a reference model in several surveys in the self-adaptation and autonomic computing fields. Therefore, the MAPE-K model is also utilised in this dissertation. However, depending on the reference, the phases of the adaptation loop are called different names. The alternative names utilised are: Collect, Analyse/Detect, Decide and Act [4,40]. Furthermore, Psaier et al. [41] present a loop for the self-healing purposes with Detect, Diagnose and Recover phases, which combine Analyse and Plan phases. Self-healing is one of the self-* properties, which form the base for self-adaptation [4]. Added to self-healing, these self-* properties are self-configuring, self-optimisation and self-protection [3,4], which can be described as follows. Self-configuring is the capability of reconfiguring by installing, updating, integrating and composing/decomposing software [4]. Self-optimisation is the capability of tuning or adjusting the parameters of the software. Self-protection is the capability of protecting against malicious attacks or unintentional usage mistakes. Lastly, self-healing is the capability of detecting and repairing from faults, errors and failures. Furthermore, self-awareness and context-awareness are essential properties for self-adaptation. Context-awareness is the capability of observing, understanding and reacting to the changes in the external (i.e. operational, environment) [42]. In contrast, self-awareness is the capability of software to know its own state and behaviour [42].

Figure 3 illustrates the MAPE-K model. The *Monitor*, *Analyse*, *Plan* and *Execute* phases form the adaptation loop and the *Knowledge* part supports these phases. In the Monitor phase, the required input information is collected by sensors. Monitoring collects input information from the execution environment, and observes the software's internal state and behaviour. This dissertation utilises context monitoring and security measures for monitoring purposes. Security

measures for monitoring purposes are the focal point. Thus, security measures are described in more detail in Sub-section 2.1.4.

The purpose of the Analyse phase is to recognise when adaptation is required. Therefore, context-awareness and self-awareness are built during the Analyse phase by combining the monitoring results. In order to recognise the adaptation need, the Analyse phase compares the currently achieved objectives with the required objectives.

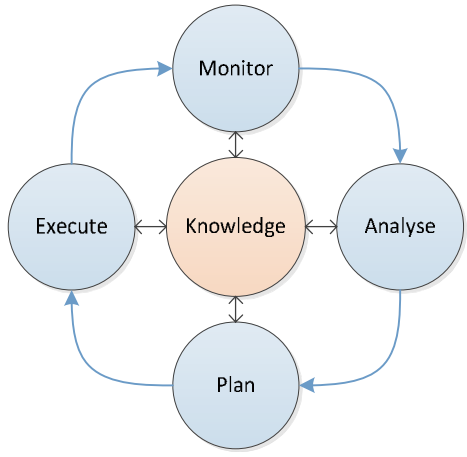


Figure 3. MAPE-K model.

The purpose of the Plan phase is to decide how required objectives are fulfilled. Therefore, the Plan recognises what has to be changed in order to achieve requirements and how to perform this change. The output of this phase is called *an adaptation plan*, which utilises self-configuring and self-optimisation. Salehie et al. divide the Plan phase into static and dynamic types [4]. Static planning means that the planning process is hard coded and cannot be modified without recompiling. In the dynamic planning, runtime modifications for the planning process are possible due to externally presented planning policies or quality definitions. The simplest way to produce an adaptation plan is to define *the ECA (Event-Condition-Action) rules* [5]. On the higher level, the alternatives are *goal policies* and *utility functions* [43]. Goal policies define the desired and undesired states, and thus, the software has to internally decide on how to achieve the desired state. A utility function calculates a value for each state, instead of only categorising desired or undesired states. Furthermore, possible trade-offs between requirements are taken into account during the Plan phase. The utility function is able to select the least bad alternative between undesired states. Hence, the utility function is the preferable selection for trade-off purposes. However, trade-off issues are out of the scope of the dissertation, and thus, utility functions are not applied in this work. Consequences of this restriction are discussed in Sub-section 4.4.

The last phase of the adaptation loop is Execute, which performs the adaptation based on the adaptation plan. The Execute phase utilises effectors, which are concrete implementations inside the adapted software to perform the specific change. In the security adaptation, effectors concentrate on causing security-related changes (i.e. changes to security mechanisms and their parameters).

The Knowledge part offers input knowledge for the adaptation loop. Knowledge is needed to describe what to monitor and how, how to interpret the monitored results, and what are the possible ways to react to the recognised changes.

2.1.4 Measures and security measuring

The security adaptation approach presented in this dissertation utilises *security measures* to obtain input information for the adaptation. The dissertation utilises the measurement terminology defined by Garcia et al. in [20]. Consequently, measures are divided into three categories that are *base measures*, *derived measures* and *indicators*. Each measure uses a *measurement approach*, which is a sequence of operations to determine a measurement result. The base measure is the smallest raw measure, which is not dependent on other measures. Hence, the base measures form the basis for all other measures. The measurement approach used with base measures is called a *measurement method*. The derived measure combines base measures and other derived measures by using a *measurement function* as the measurement approach. Indicators are the most complex measures, which combine base measures, derived measures and other indicators by using the measurement approach, called *analysis models*. Figure 4 illustrates the structure of measures and their dependencies (i.e. arrows direct to measures, which utilise other measures).

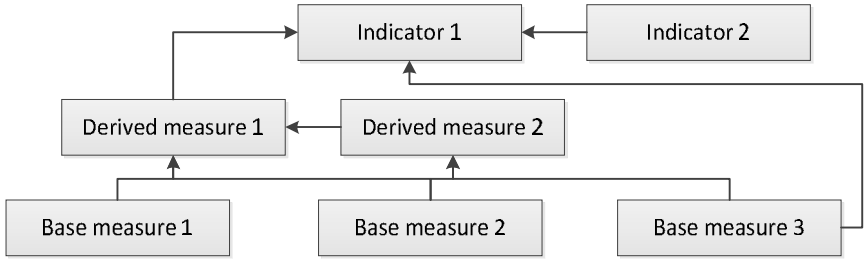


Figure 4. Structure of measures.

In some sources base measures are called direct measures and derived measures and indicators are called indirect measures [44,45]. Moreover, the term *metric* is widely used – especially in the security-measuring context. However, in this dissertation terminology from Garcia et al. [20] is utilised because it is based on an alignment made from several software measuring and metrology standards. Hence, the term security measure refers to base measures, derived measures and

indicators for security. Security measure does not refer to security countermeasures, which are called security mechanisms.

Each derived measure and indicator depends on base measures, which are measured by means of a measurement method. In the runtime security adaptation, the measurement method has to be a concrete code snippet, which implements a measurement method for the runtime purposes. In this dissertation, these code snippets are called *monitoring probes*, which are presented with a Unified Modelling Language (UML) component symbol in order to emphasise their existence in the software.

From the measuring viewpoint security is not a single measurable attribute. Security is the composition of security objectives, and thus, security measuring has to reach security from different objective viewpoints. Wang and Wulf present a decomposition approach to dividing security into smaller and more measurable parts in [46]. The authors present decompositions for confidentiality, integrity, authentication and non-repudiation. Savola et al. utilise the decomposition approach in [47] to define decompositions for authorisation and availability. From the measure structure viewpoint (c.f., Figure 4), decomposition has to be continued until the leaf nodes are found, which are measurable with base measures.

Bottom-up and top-down security measure definitions are presented in [48]. Decomposition is a top-down method to generate security measures. In the top-down method, the measure definition starts from the security objective and proceeds to the required bottom-level base measures. In contrast, the bottom-up measure definition starts from the bottom level to seek what base measures are available and defines derived measures and indicators based on the available information. This dissertation utilises the bottom-up approach to define security measures. Developing security measures by means of a top-down approach produces a comprehensive set of measures. However, security adaptation architecture and the knowledge base are the focal point of this work. Therefore, it is reasonable to continue when the sufficient set of security measures is available – instead of defining the most extensive measure set.

2.1.5 From quality variability to quality adaptation

Svahnberg et al. define software variability as the ability of the software system or artefact to be efficiently extended, changed, customised or configured [49]. Variability makes it possible to reuse existing software components and architectures, and in addition, it supports delayed decision making during the software design phase. Variation points are the points at which changes caused by variations take place [50]. Examples of the mechanisms to achieve variability are inheritance, extensions, configuration, template instantiation, generation, inclusion or omission of elements, and the selection of versions of elements [50,51]. In order to utilise coherent terminology, this dissertation utilises the terms self-configuration and self-optimisation to refer to runtime variation mechanisms.

Binding time defines when the variation takes place [52]. Consequently, the latest possible binding time is at runtime [49,52]. In the context of this dissertation, the variability occurring at runtime is called adaptation. In [52] Niemelä et al. define the quality variability model. Added to binding time, the model defines *scope*, *importance* and *dependencies on other quality attributes* for variability. The *scope* declares how widely the variation is able to have an effect (i.e. on the family, product, service or component levels). The *importance* defines how quality variation can take place by categorising the variations into high, medium and low categories. The high category means that quality cannot be lowered in a normal operation. Nevertheless, the quality can be lowered temporarily if it is mandatory for the survival of the whole system. Consequently, the importance concept is essential when analysing trade-offs. Finally, the *dependencies on other quality attributes* concept declares connections between quality attributes. For security, dependencies on performance and reliability exist (i.e. security mechanisms have to be reliable and mechanisms consume resources). The quality variability model presented above states that dependencies exist between quality attributes. However, in security internal dependencies also exist between security objectives. For instance, adapting authorisation may affect the utilised identification scheme.

2.2 Related work

2.2.1 Security adaptation approaches

This section summarises existing security-adaptation approaches. The MAPE-K is commonly utilised as a reference model in different adaptation approaches, and thus, elements from the MAPE-K model are emphasised in this section.

Elkhodary et al. surveyed four security adaptation approaches in [12] – namely Extensible Security Infrastructure [53], Strata Security API [54], The Willow Architecture [55] and the Adaptive Trust Negotiation Framework [56]. As the final conclusion, authors notice that any of these approaches support all security objectives but concentrate on specific and pre-selected objectives. Moreover, maintainability and reusability were recognised as a challenge.

Russello et al. propose the Architectural Approach for Self-managing Security Services in [57]. The adaptation approach utilises Event-State-Condition-Action (ESCA) policies. The approach separates the application logic to an application layer, while adaptation elements and security mechanisms are located on a middleware layer. An ECSA policy manager selects policies and the set of security mechanisms, which are adapted based on the selected policy. The following context monitoring services offer contextual information: Trust level, threat level, availability monitor and bandwidth monitor. However, the internal design of these services is not described. The utilisation of the separation of concern principle ensures that the architecture is clearly described. The Context sub-system performs monitoring and analysing, while the ECSA policy manager performs planning and execution. The required knowledge is presented inside the Context monitoring

services and ECSA policies. The ECSA policy manager is validated in [58,59]. However, the validation does not contain a security-adaptation viewpoint.

The Software Framework for Autonomic Security in Pervasive Environments consists of monitoring, analysing and responding modules with their mutual interactions [15]. The monitoring module observes security-related events. The analysing module receives these events and suggests a high-level security action in order to reconfigure. Consequently, the module combines analysing and planning phases. Finally, the responding module delivers reconfiguration actions for the implementation specific sub-systems. The approach contains a support module, which offers a profile database for other modules. The analysing module is able to get information from the database in order to perform decision making. However, the content of the database is not described.

Hulsebosch et al. present Context Sensitive Adaptive Authentication in [14], the initial version of the approach is presented from the access control viewpoint in [60]. The approach utilises the user's context information (i.e. location and time) for the adaptation. The idea is to approximate the authentication confidence with the probability of the user being at a certain location in authentication time. Hence, the authors present a fusion algorithm that calculates probability values for the user's location. The main focus of the approach is the utilisation of the context information for the adaptive user authentication. Therefore, the monitoring part concentrates on context monitoring. Moreover, the analysing part is partially covered by calculating location probabilities. However, the authentication-level decision is left for the application. The approach does not concentrate on the planning phase or contain a separated knowledge base.

Genetic Messaging-Oriented Secure Middleware (GEMOM) contains self-healing and adaptation features [61]. Security adaptation in GEMOM is performed by means of an Adaptive Security Manager (ASM) [62]. The ASM contains elements for monitoring, analysing and adaptation. Thus, the adaptation element combines plan and execute phases. In the GEMOM the monitoring is the most emphasised part, which contains anomaly detection, Quality of Service (QoS) monitoring and security measuring. From these monitoring techniques, security measuring is described in more detail in [47]. Analysing and planning are performed in separated components. However, the content of these phases is not described. The approach contains a database in order to offer input for the analyses but the content is not described.

The most recent survey in this field is presented by Yuan and Malek in [13]. The survey compares over 30 self-protecting approaches in a tabular form. The survey reveals that almost all approaches cover the monitoring phase, but the planning phase is not covered as extensively. Moreover, it is visible that only few approaches take adaptation requirements into account from design- and runtime viewpoints. Lastly, the survey denotes that there is a need for architecture-based approaches that support generality and scalability. Actually, only two approaches were found that cover the whole adaptation loop and take architecture aspects into account. These approaches are Rainbow [63] and SASSY [64]. Rainbow concentrates on performance, availability and cost aspects. SASSY is intended to be

generic in order to maintain functionality and QoS requirements by means of adaptation. Consequently, these approaches are not directly intended for security-adaptation.

2.2.2 Security ontologies

In this work, knowledge for security adaptation is formed by means of a security ontology. Gruber defines ontology as follows: “An ontology is an explicit specification of a conceptualisation” [65]. Afterwards, Zhou defines ontology as a shared knowledge standard or knowledge model, defining primitive concepts, relations, rules and their instances, which comprise topic knowledge. It can be used for capturing, structuring and enlarging explicit and tacit topic knowledge across people, organisations, and computer and software systems [66]. On the other hand, Chandrasekaran et al. emphasise the importance of ontologies by their capability to enable knowledge sharing and coherent reasoning [67].

The original publication PI compares four security ontologies from runtime and measuring applicability viewpoints. These ontologies are Taxonomy of Information Security in Service-Centric Systems [18], Security for DAML Web-services [68,69], Security Ontology for Annotating Resources [70] and Security Ontology by Tsoumas et al. [71]. The comparison revealed that these ontologies are intended either for design-time usage or for service discovery and matchmaking purposes. Moreover, Blanco et al. compare and analyse security ontologies in [72,73]. The authors conclude that most of the existing security ontologies concentrate on describing the particular security area or domain, for instance access control or auditing. Furthermore, all developed security ontologies are not available [73]. Nevertheless, Blanco et al. have found few general purpose security ontologies, which are available – namely ontologies from Denker et al. [68], Kim et al. [70], Fenz et al. [74] and Herzog et al. [21]. Fenz et al. have published their work in several publications, for example [74,75]. These papers include the information-security viewpoint but the ontology contains threats and controls for physical security purposes also, for instance fires and safety doors. The authors’ focus is clearly on risk management, especially the business continuity aspects, which in turn do not lend themselves to the scope of this dissertation. In contrast, the ontology from Herzog et al. [21] concentrates purely on information security and it contains more concepts than other security ontologies. Hence, the ontology from Herzog et al. is utilised in this dissertation as a starting point to build a security ontology for adaptation purposes. In this work, the ontology from Herzog et al. is abbreviated with an OIS (Ontology of Information Security).

The OIS categorises the concepts around risk-management terminology: assets, threats, vulnerabilities, mechanisms and security objectives (terms shown in Figure 2), which form high-level concept classes for the ontology. Moreover, the ontology is not only a class hierarchy but connecting properties between classes are also presented, for example threats threatening the particular security objective. However, the OIS does not contain security-measuring terminology. Conse-

quently, this dissertation extends the OIS for security measuring and adaptation purposes.

Figure 5 defines a graphical notation utilised in the dissertation to present ontologies. Rectangles refer to concepts, which are ontology classes and instantiated classes. In the figure, *Concept1* is a concept and *Concept2: instance1* is an instance created from Concept2. Relationships are depicted by means of arrows, which present a reading direction and name (property name). In addition, arrows contain cardinality information. Consequently, the figure defines that Instance1 (instantiated from Concept2) relates to one or more of Concept1.

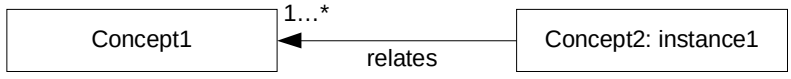


Figure 5. Graphical notation utilised in the dissertation.

Naturally, these graphical presentations are not suitable for validation purposes when ontology has to be in a machine-readable form. OWL [76] is utilised for this purpose, which is a de-facto standard to present ontologies in machine-readable form. OWL is based on the Resource Description Framework (RDF) [77] and RDF Schema (RDF-S) [78].

3. Research

This chapter gives an overview of the research contributions of the dissertation. More details are given in the original publications that are listed at the beginning of the dissertation and attached at the end. The research problem is divided into the architecture and knowledge base objectives in Introduction. These objectives form the main topics for the dissertation. The research to fulfil the architecture and knowledge base objectives was made in parallel because the architecture utilises knowledge, and the structure of the knowledge has an influence on the architecture.

The architecture objective set in the Introduction states that the architecture has to contain the whole adaptation loop and, in addition, provide a means of monitoring the achieved security level in order to trigger adaptation. Consequently, the architecture research contains architectural elements, as well as their mutual behaviour and deployment, in order to achieve security adaptation. The presented adaptation architecture is designed to support the security measures for monitoring purposes to recognise the adaptation need.

The knowledge base forms the second research topic for the dissertation. The knowledge base objective requires developing a knowledge base, which is applicable at design-time and runtime alike. Furthermore, it is required that the knowledge is presented in a reusable and extensible manner. The knowledge base is created by using an ontology, which ensures that knowledge is retrievable both for software architects at design-time and applications at runtime.

The research utilises a constructive research approach by means of five cases. Sub-section 3.1 briefly describes the cases. Sub-section 3.2 then summarises the contributions to the architecture objective, and lastly, Sub-section 3.3 describes contributions from the knowledge base point of view.

3.1 Cases

The cases that form this dissertation are summarised in **Table 5**, where case objectives and experiences are presented from the perspective of this work. Based on Runeson et al. [79], the case objective defines what is expected to be achieved in the case. In Table 5 cases C1–C5 are mapped to the original publications PII–PVII. All cases contribute to the knowledge base, and C2, C3 and C5 also contribute to

the architecture. In addition, the cases contain both design-time and runtime viewpoints, which is one novelty claim stated in Sub-section 1.2.

Table 5. Case objectives and experiences.

Case/Pub.	Case objective	Experiences
C1/PIV	Investigate how to retrieve quality knowledge from the ontology at design-time.	The software architect is easily able to retrieve quality knowledge from the ontology when a tool support exists. Furthermore, separating quality measures from other quality knowledge is a reasonable solution.
C2/PII and PIII	Develop an architecture that utilises security measures and ontologies for adaptation purposes.	The main functionality of the adaptation architecture is in place. However, the content and the order of the adaptation phases need enhancements. Similarly, the ontology contains the main concepts but a finer granularity is needed.
C3/PV	Evaluate the first version of the ISMO for design-time and runtime purposes. In addition, the objective is to develop architecture parts for measuring purposes.	The case proves that the architect is able to find applicable security mechanisms and base measures from the ISMO (the ISMO ontology is described in Sub-section 3.3.1). Furthermore, the ISMO offers a means of retrieving analysis models in order to calculate security-level indicators at runtime. The designed architecture elements ensure that only components for base measures are implementation-specific, while other measuring-related components are generic and reusable.
C4/PVI	Extend the ISMO with new measures.	It is possible to extend the ISMO with new measures. The case shows that the ISMO is able to present design-time knowledge for runtime adaptation purposes.
C5/PVII	Develop the architecture further based on previous experiences – applying the MAPE-K reference model. Utilise knowledge from the ISMO in all phases of the adaptation.	Utilising the MAPE-K reference model for the security adaptation ensures that the adaptation phases are separated. In addition, a distinct knowledge base supports knowledge modifications and reduces hard-coded adaptation knowledge. Lastly, the case shows the knowledge required during the adaptation phases.

The SMEPP case is the first case (C1) presented in PIV. The SMEPP case is intended to validate the knowledge-based quality-driven architecture design and evaluation method that utilises ontologies as one knowledge source. The case concentrates purely on software design-time. Therefore, the case shows how the architect utilises quality knowledge from ontologies. The case utilises the reliability ontology [19] because it contained more content than the security ontology at that time. The SMEPP middleware has several reliability requirements and the architect searches reliability measures from the ontology to reveal if requirements are not fulfilled. The reliability ontology is presented in the OWL format and a tool

called the Quality Profile Editor (QPE) is used to search quality measures from the ontology. The case showed that an appropriate tool facilitates retrieving knowledge from the ontology at design-time. In addition, the case indicates that separating quality measures and a means to achieve the particular quality is a reasonable decision. Consequently, it is possible to update and extend both knowledge areas without knowing the other.

The second case (C2), the Smart Greenhouse, contributes to both the architecture and knowledge base objectives. The case is presented in PIII. However, the initial vision of the architecture has already been presented in PI, and is further developed in publications PII and PIII. The purpose of C2 is to develop a security-adaptation architecture that utilises security measuring and ontologies. The case contains a miniature greenhouse, and a gardener utilises greenhouse devices by means of his mobile device. The security adaptation occurs in the smart space application located in the gardener's device. The gardener is able to retrieve sensor information from the greenhouse and control the actuators. The greenhouse contains a shopping area for customers. The number of customers in the greenhouse area is calculated and this value is further utilised to calculate risk levels. In other words, arriving customers and their mobile devices are recognised as threats from the gardener's application viewpoint. The case utilises risk-based security measures for confidentiality and integrity, which are stored into the security ontology. In addition, the ontology describes the security mechanisms that support these security objectives. It is notable that PIII utilises the term *security concept* instead of *security mechanism*. The gardener's application utilises security measures and searches for adaptation means from the ontology. Consequently, the case fulfils its objectives. The developed security adaptation architecture follows a loop structure, which is visible, especially in PII. However, the granularity of the different phases in the adaptation loop varies (c.f. Figure 2 in PII). On the one hand, some phases are trivial (e.g. a phase called *Retrieve supporting mechanisms from ontology)*, and on the other hand, some phases contain lot of functionality (e.g. the *Measuring and Reasoning* phase). Consequently, the architecture structure is further developed in the following cases. The security ontology utilised in the case contains all the necessary concepts to achieve the first version of the security adaptation. However, the amount of security knowledge is not adequate and the measures are described in a simple form.

Case C3 evaluates the first version of the ISMO and develops an architecture for measuring purposes. This is the first case that utilises the ISMO in its current form that reuses existing security and measuring ontologies in order to provide an extensive set of concepts. Details of this case are presented in PV. The case takes place in a smart home environment and the smart space application with the adaptation feature runs in a mobile device. In the smart home, a user is able to perform actions of varying criticality, which cause a need to adapt security levels. Security adaptation in this case is implemented for the authentication security objective, in a situation where authentication is based on passwords. At design-time, the architect searches base measures for password authentication from the ISMO. The developed architecture elements (c.f. Figure 7) ensure that only the

software components for the measurement methods of the selected base measures have to be implemented into the smart space. In other words, other measuring-related components are reusable as such. At runtime, the application searches knowledge from the ISMO. Hence, the required analysis models are retrieved from the ISMO in order to calculate the authentication-level indicator for different situations. The case shows that the structure using base measures, derived measures and indicators to present security measures supports reusability and ensures that the measures are presented in a machine-readable form. The authentication levels, required for different actions, are hard coded in this case. Moreover, the Monitor component calculates the security levels (c.f. Figure 6 in PV). These decisions were reasonable during the case implementation but the enhanced version is created in C5.

Case C4 concentrates on design-time aspects and shows how to extend the ISMO with new knowledge. PVI presents the case in more detail. In the case, the architect designs software with the security-adaptation feature by utilising the design steps presented in PVI in Figure 2. After the architecture design, the RAP method is utilised to predict reliability values for alternative authentication mechanisms (i.e. authentication variants). The reliability values are presented by using the probability of failure (pof) values [80]. Initially these pof values are intended for the software architect to support design-time decision making. However, the case extends the ISMO in a way that pof values can be offered for runtime purposes. The extension contains instances for the following concepts: *Attribute*, *BaseMeasure* and *MeasurementMethod* (c.f. Figure 5 in PVI). Therefore, all the instances required to add a new measure are included. The case proved that new measures can be added into the ISMO. In addition, the structure of the measures is not security specific, and measures for different qualities can also be added. Reliability values offer knowledge from design-time, which can be used in the Analyse phase as one factor in a security-level calculation.

Case C5 (the SUM-SS pilot) is the last case, which is presented in PVII. In addition, the general-level presentation of the SUM-SS pilot is given in the Sofia brochure [81]. The SUM-SS pilot contains four smart spaces: a smart personal space, a smart home, a smart office and a smart city. The owner of the smart home acts as an end user in this case. The home owner moves between smart spaces and simultaneously controls devices located in the home smart space. The smart space application, with the security adaptation features, is running in the end user's mobile device and authentication is the adaptive security objective. Thus, this case has similarities to C3. However, the case contains the latest versions of the adaptation architecture and the ISMO. Consequently, the case takes into account experiences gained in previous cases. In addition, the case occurs in a more heterogeneous environment and the application utilises information distributed for different smart spaces. The used security-adaptation architecture conforms to the MAPE-K reference model. Hence, the case contributes to the structure of the security adaptation architecture as do the others. However, the case contributes behaviour and deployment descriptions to the architecture, which are not considered as much in previous cases. In the Plan phase, the case utilises

AnalysisModels in order to find attributes, whose adaptation is able to change the achieved security level (c.f. Figures 9 and 14 in PVII). Furthermore, the adaptation phases are connected to the required knowledge from the ISMO. The case proves that utilising the MAPE-K reference model produces the security adaptation architecture with common and reusable adaptation phases. Separating the knowledge from the adaptation loop ensures that the smart space application does not need hard-coded analysis models or planning rules.

3.2 Security adaptation architecture

This sub-section describes contributions to the security-adaptation architecture. The architecture contains the elements, their mutual behaviour and deployment in order to achieve security adaptation. Next, Sub-sections 3.2.1, 3.2.2 and 3.2.3 describe the proposed architecture from structural, behavioural and deployment viewpoints, respectively.

3.2.1 Structure

The structure of the security adaptation architecture is developed iteratively in cases C2, C3 and C5. Due to iterative development, the architecture structure from the last case forms the main contribution. However, the measuring-related architecture elements have already been recognised in C3 and fine-tuned in C5.

The architecture structure is presented in Figure 6. The structure conforms to the MAPE-K reference model. Consequently, the *Monitor*, *Analyser*, *Planner* and *Executor* components play a key role in the structure, that is, the architecture applies the whole MAPE adaptation loop for the security adaptation and defines each phase separately. The *knowledge* is offered from the OWL-formatted ontology, the ISMO, which is described in Sub-section 3.3. The ISMO is connected to the Monitor, Analyser and Planner components, which utilise knowledge from it. PVII defines the knowledge required in these adaptation phases. Moreover, the component descriptions below summarise the knowledge retrieved from the ISMO. The existing security adaptation approaches do not define knowledge and its utilisation in the MAPE loop. Hence, this dissertation defines the knowledge base, and in addition, describes what knowledge is needed in the adaptation phases.

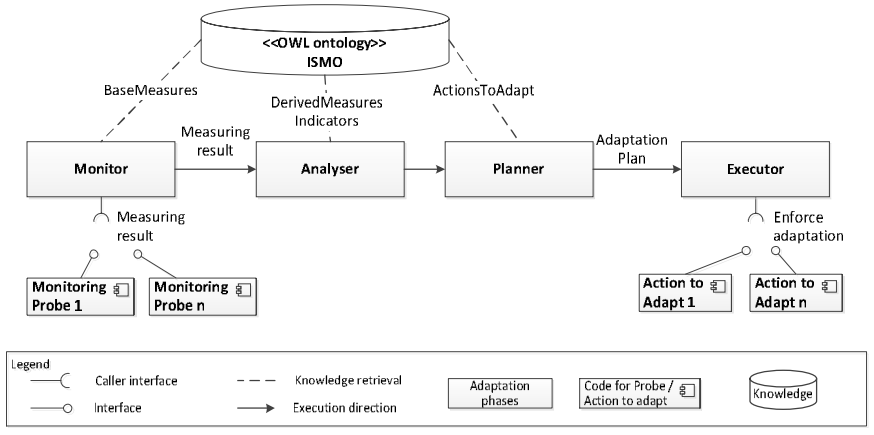


Figure 6. The structure of the adaptation architecture.

The *Monitor* component is connected to the *Monitoring probe* components, the Analyser and the ISMO. From the ISMO, the Monitor component retrieves the *base measures* to use. Thus, only measures for required *security objectives* and utilised *security mechanisms* are used. Each base measure has its own measurement approach (i.e. *measurement method*) that describes how to perform the measurement, which is available from the ISMO. The Monitoring probe components are concrete code snippets that implement the measurement methods. Hence, Monitoring probes are reusable building blocks for performing measuring. The Monitor component requests measuring results from the selected Monitoring probe components. The proposed solution utilises security measures to monitor the achieved *security level*. This makes it possible to define generic and implementation-specific parts for the security-level monitoring, which is novel compared to the existing approaches as stated in Sub-section 1.2. Figure 7 presents these generic and implementation-specific parts. Initially, this structure is presented in PV, where the separation of measurement methods (i.e. monitoring probes) and the Monitor component is presented. However, the security-level indicator calculation is allocated for the Analyser component in this final architecture, as presented in PVII and visible in the figure.

The *Analyser* component is called by the Monitor component. Figure 7 shows the internal components of the Analyser component to calculate the *security level indicator*. The Analyser component retrieves *derived measures*, *indicators* and related measurement approaches (i.e. *measurement functions* and *analysis models*) from the ISMO. *The Analysis model parser* component parses *rules* from analysis models, which are utilised in the *Base measure combiner* component to calculate *the security-level indicator*. In other words, the retrieved measurement approaches act as equations to calculate results for derived measures and indicators by combining results from base measures, as described in Sub-section 2.1.4. Hence, the Analyser component is able to calculate the achieved security level indicator in

order to recognise the adaptation need. The whole Analyser component is located in the generic part, as visible in Figure 7. Therefore, the Analyser component does not change when the utilised Monitoring probes change, which supports reusability and extensibility. This advantage is achieved due to the utilisation of security measuring and a three-level measuring structure.

Afterwards, the Analyser component compares the achieved and required security levels and calls the Planner component if the required security has not been achieved. C2 and C3 utilise hard-coded security levels to set the required securities. PII defines the context information that affects security. Therefore in C5, the Analyser component uses context information to deduce the required security levels (c.f. Figure 8 in PVII). Thus, the Analyser component produces both the required and achieved security levels by utilising the monitored information. Nevertheless, deciding the required security objectives and levels at runtime is not the focal point of this dissertation.

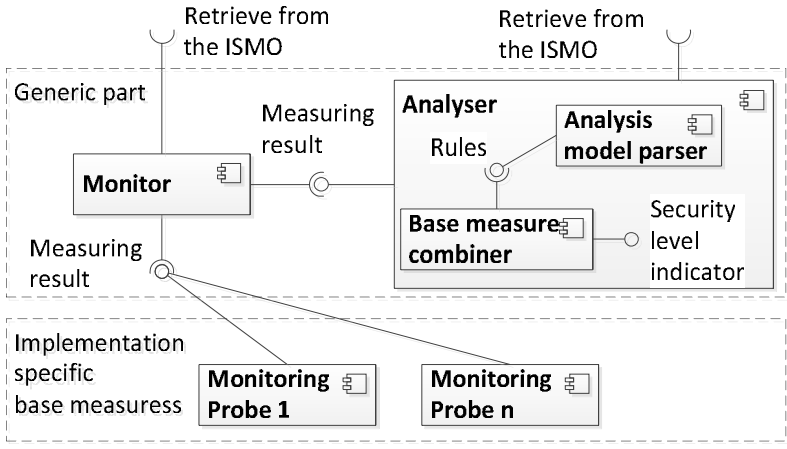


Figure 7. Generic and implementation-specific parts for security-level monitoring.

The purpose of the *Planner* component is to create *an adaptation plan*. The component is connected to the ISMO in order to retrieve alternative security mechanisms or attributes to achieve the required security. The adaptation plan is either defined at design-time, decided at runtime based on knowledge from the ISMO, or in the worst situation, instructions on how to proceed are requested from the user. These alternative ways to create the adaptation plan are described in PVII and summarised in the Behaviour Sub-section 3.2.2.

The *Executor* is the last component in the adaptation loop. The purpose of the Executor component is to enforce the adaptation plan received as an input from the Planner component. Thus, the Executor component is connected to the *Action to Adapt* components, which are concrete implementations for adapting security.

Therefore, the Action to Adapt components are security mechanisms intended to use, or alternatively, implementations that modify the attributes of security mechanisms.

3.2.2 Behaviour

This section summarises the mutual behaviour of architecture components. The behaviour is presented by means of sequence diagrams, which contain components from Figure 6.

Figure 8 depicts the first part of the sequence diagram, where the Monitor and Analyser components play a key role at the beginning of the adaptation loop. The adaptation loop starts from the *Monitor* component. Firstly, the Monitor component retrieves the *applicable base measures* from the ISMO. As mentioned earlier, the selection of base measures depends on the *required security objectives* and the *security mechanisms* that are utilised to fulfil those objectives. For example, it is not reasonable to utilise measures for integrity if that security objective is not required. When the base measure selection is ready the Monitor starts the *Monitoring probe* components, which in turn start to offer *measurement results*. The Monitor component delivers the measured results to the *Analyser* component. The first time, the Analyser searches for the *derived measures*, *measurement functions* (measurement approach for derived measures), *indicators* and *analysis models* (measurement approach for indicators) from the ISMO. For clarity reasons the sequence diagram in Figure 8 shows only the retrieval of analysis models. By means of the analysis model the measurement results from the Monitoring probes are composed to the *security-level indicator* as described in Figure 4. The security-level indicator produced with the analysis model is compared to the *required security*. If the required security is achieved adaptation is not needed and the execution returns to the Monitor component. Otherwise, the *Planner* component is called. In Figure 8, *Measurement1* does not cause an adaptation need but *Measurement2* reveals that the achieved security level is not sufficient and the Analyser component calls the Planner component.

PVII and C5 utilise context information in order to avoid hard-coded security requirements. Hence, context information is also monitored and the results are delivered for the Analyser component. The Analyser component calculates the required security and compares the required and achieved security levels in a similar way to that presented in Figure 8. In other words, context information is treated similarly to the security measurement results. However, the means of performing context monitoring are not the focal point of the dissertation; more details on context monitoring are described in [25].

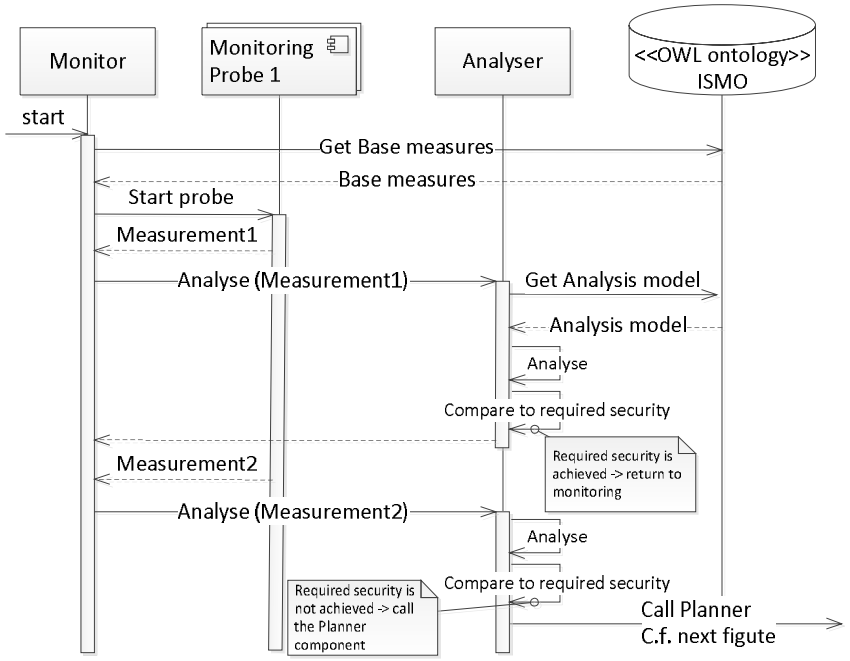


Figure 8. Sequence diagram part 1.

Figure 9 presents the behaviour when the Analyser component calls the Planner. The heterogeneity of smart spaces causes challenges for the Planning phase. The *adaptation plan* that is sufficient in one situation is not automatically applicable in another situation. Consequently, the adaptation plan's applicability has to be checked in each situation. In PII and PIII adaptation architecture contains a separated phase called Control analysis that checks the mechanism's applicability for the current environment. However, in this final architecture this functionality is an internal functionality of the Planner component. PVII enumerates three alternative ways to create an adaptation plan: i) The smart space application utilises a predefined adaptation plan; ii) The application searches for alternative security mechanisms or attributes to adapt from the ISMO; iii) The application asks the user how to proceed. These alternatives are presented inside the Alternatives frame in the sequence diagram and separated from each other with a dashed line.

In the first alternative, a *predefined adaptation plan* is utilised. For example, when the security objective of confidentiality is not achieved the predefined adaptation plan can be "start to use encryption". Thus, the Planner component selects the adaptation plan and delivers it to the Executor component, which in turn calls the related *Action to Adapt* component. From the previous example, the Action to Adapt component is a concrete implementation that enforces the utilisation of the particular encryption library, whose handshake procedure takes care that both communication sides start to use encryption and negotiates the required parameters.

The final step of the adaptation is to return to the Monitor component and continue the adaptation loop.

In the second alternative, the Planner component utilises the *knowledge from the ISMO* in order to create the adaptation plan. On the one hand, it is possible to search for *alternative security mechanisms* based on the required security objectives. On the other hand, it is possible to get the utilised analysis model from the ISMO and find *attributes* that have affected the achieved security level, and adapt those attributes. How these attributes can be found is presented in PVII in Figure 9. Based on the knowledge retrieved from the ISMO, the Planner component creates the adaptation plan and calls the Executor component, which enforces the plan. Again, the execution returns to the Monitor component.

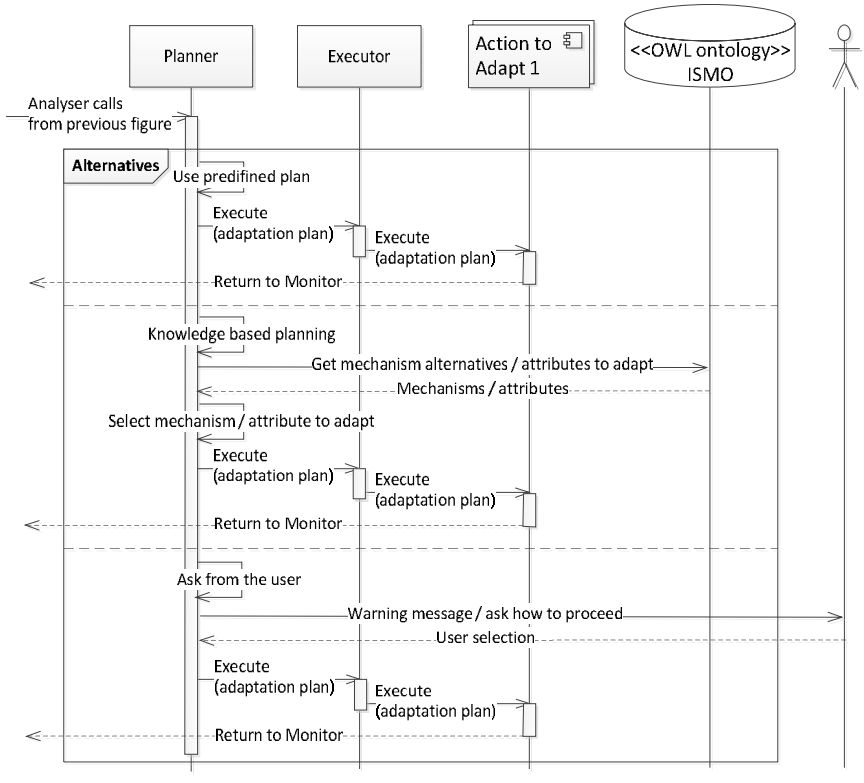


Figure 9. Sequence diagram part 2.

The third alternative way to create the adaptation plan is intended for situations when the above described means are not sufficient to decide how to proceed. Therefore, the only possibility is to ask for instructions from the end user. Thereafter, the sequence proceeds similarly as described in alternatives one and two.

It is not always possible to create an adaptation plan that satisfies all security and other quality requirements. For instance, the security mechanism can require computation resources, which cannot be allocated for it. Or alternatively, the adaptation plan can propose a security mechanism (e.g. multifactor authentication) that reduces usability. Moreover, the smart space devices may define conflicting adaptation plans. It is mandatory to take these trade-off and matchmaking issues into account during the Plan phase. Currently, the Planner component utilises *goal-orientated decision making*, that is, the context sets the *required security objectives* and *levels* (goals) and the Planner component finds a configuration that satisfies the goal. However, taking trade-offs and matchmaking into account requires more sophisticated decision making in the Planner component, for example utility functions and negotiations between devices. The selection of a decision-making algorithm is the internal design decision of the Planner component, which is not restricted in the presented architecture. This future research topic is discussed in Sub-section 4.4.

3.2.3 Deployment

The last part of the security adaptation architecture is the deployment viewpoint. The deployment diagram is depicted in Figure 10. In the figure the smart space is abbreviated to SS. RIBS is the RDF Information Base Solution, which is described in Sub-section 2.1.1. However, the architecture is also applicable for other smart space infrastructure implementations. For instance, C2 utilised Smart-M3.

The adaptation loop (i.e. the Monitor, Analyser, Planner and Executor components) and the ISMO are contained in a device where the application is utilised and where the adaptation is intended to occur. In other words, the adaptation is intended to occur from a Device1 perspective. Performing the adaptation from the Device2 or RIBS perspective requires that the adaptation loop and the ISMO are located in those devices. Similarly, in the smart space where all devices utilise security adaptation, the adaptation loop is located in all devices. Consequently, Figure 10 presents one possible deployment instantiation. The M*onitoring probe* and the *Action to Adapt* components are distributed over the whole smart space. For instance, C2 counts customers by using a Monitoring probe (recognition algorithm for video stream) outside the gardener's device and this monitored data is utilised in the gardener's device to calculate the threat level indicator from the gardener's perspective. On the other hand, the adaptation action that changes the utilised communication protocol has an effect on both sides of communication. The example in Sub-section 3.2.2 contains the following adaptation plan "start to use encryption". A device that initiates this adaptation plan contains an *Action to Adapt* component to enforce the utilisation of the particular encryption library. For the other side of communication this adaptation looks like a normal part of the encryption protocol handshake. Thus, conceptually the Action to Adapt component in the other side of communication is a part of the encryption library. The adapting

device is able to recognise if the communication channel is not encrypted. Hence, the adaptation does not cause additional trust requirements between devices.

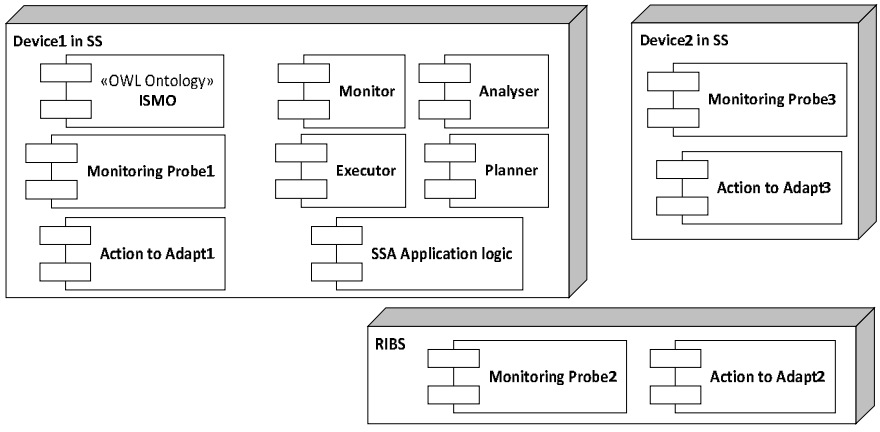


Figure 10. Example deployment of the security-adaptation architecture.

The justification to centralise the *adaptation loop* and the ISMO is two-fold. Firstly, the adaptation is intended to occur from the Device1 viewpoint. Thus, the security adaptation analyses security and plans adaptation for Device1 purposes. In other words, the security consequences of the monitoring results are analysed in order to reveal how those affect Device1. Similarly, Device1 makes a decision on how to adapt based on its own preferences and capabilities. Secondly, trust requirements dictate this deployment because it is more preferable to trust own computation and storage capabilities instead of unknown external devices. In public smart spaces, it is particularly challenging to know the trustworthiness of other devices. Thus, the centralisation ensures that the trust level of the content of the ISMO and the results of the Analyser and Planner components do not need to be evaluated. In contrast, storing the ISMO in another device makes it possible that an attacker modifies its content. In the same way, distributing Analyser and Planner components creates an attack possibility. Naturally, the trustworthiness of the monitoring probes is also an important research issue in the future. Therefore, the analysis model has to take trust levels into account when the monitoring probes are located in other nodes.

Although the adaptation loop and the ISMO are centralised in a device where the adaptation is intended to occur, it does not restrict the decentralisation of smart space applications and devices in the smart space. Decentralising the adaptation loop and the ISMO requires reasonable justification of the possible advantages and the evaluation of appearing threats. However, the performed use cases have not indicated any demands towards decentralisation from the viewpoint of the adaptation loop and the ISMO.

3.3 Ontology as a knowledge base

This sub-section describes contributions to the knowledge base objective. Sub-section 3.3.1 describes the structure of the ISMO, 3.3.2 shows the design-time usage of the ISMO and runtime usage is presented in Sub-section 3.3.3.

3.3.1 Structure

Security adaptation requires: i) Security knowledge, ii) Measuring knowledge and iii) Context knowledge. Security knowledge defines security objectives, mechanisms, threats and how they are related. Thereafter, measuring knowledge describes attributes and how to measure them. Lastly, context knowledge describes the smart space and the role of data, users and actions within the smart space. These three knowledge areas are presented in Figure 11, which is originally presented in PVII. Moreover, the figure shows the relationships between these main concepts. The purpose of the concepts and relationships are described in more detail in Sub-section 3.1 of PVII.

Case C2 utilises a security ontology that contains a minimal set of security and measuring terminology. In other words, the security ontology contains concepts that are specifically required in C2. Starting from PV security ontology is called the ISMO. In PV, the ISMO is composed of the Software Measurement Ontology (SMO) [20] and Ontology of Information Security (OIS) [21]. This ontology reuse makes it possible to create a wide ontology, whose concepts have already been validated in previous scientific publications. The ISMO is a novel contribution since it contains both security- and measuring-related knowledge in an ontology form. This kind of knowledge composition has not been presented before. Moreover, the ISMO is applied as the knowledge base for the security adaptation in this dissertation. The research made in PV revealed that there are only few concept-level mappings between security and measuring-related terminology. However, the additional mappings appear when security measures are instantiated into the ontology. Taking an example from the figures presented in PV, password age is an *attribute* of the password credential and related to the brute-force attack threat. However, these relations only appear when an attribute instance (password age) is created. It cannot be defined that all credentials have an attribute and that attribute relates to a threat (i.e. concept-level mappings). Consequently, new mappings are created to the ISMO in cases C4 and C5 when new measures are added. The ISMO is an OWL-formatted ontology and available from the web[1]. The OWL file is produced by means of the Protégé ontology tool [82].

Context knowledge is stored in the Context Ontology for Smart Spaces (CO4SS), which is described in detail in [83]. From the security perspective, the

[1] https://sofia-community.com/projects/sontologies/

CO4SS is developed in PII by defining the security-related context information and mapping it to Situational, Digital and Physical context levels. This security-related context knowledge is utilised in C5 to define the required security levels for different situations.

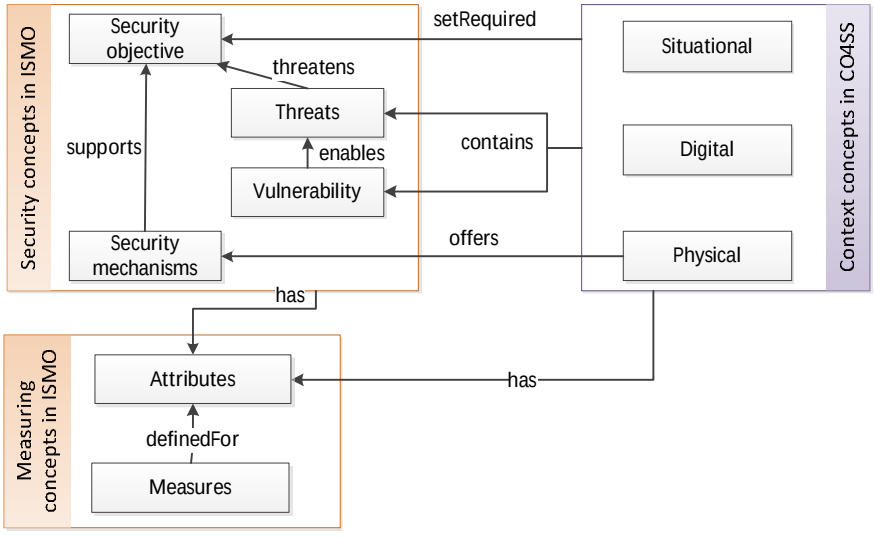


Figure 11. The structure of the main concepts presented in PVII.

3.3.2 Ontology usage at design-time

The knowledge base objective states that the knowledge base has to be able to offer knowledge for design-time and runtime purposes alike. However, knowledge requirements for design-time and runtime usage are different. In addition, design-time and runtime users of knowledge are not the same. At design-time, a software architect is the main stakeholder that retrieves knowledge from the ISMO, whereas the smart space application retrieves knowledge at runtime. Initially, the architect has a *set of security objectives* required from the designed application. From the ISMO the architect is able to retrieve *security mechanisms* that support the required *objectives* (c.f. Figure 12). Furthermore, the ISMO contains a means to measure the *security level* of the particular *security objective*. That is, each security objective has an *indicator* for the security level. The indicator is calculated by means of an *analysis model*, which depends on the selected security mechanism. In the same way, each analysis model uses the specific set of *base measures*, as visible in Figure 12. It is notable that analysis models use base measures via other analysis models and derived measures as depicted in Figure 4. However, these are not shown in Figure 12 for clarity reasons. Moreover, it is notable that the same base measure can be utilised in several analysis models. The architect's

task is to design and implement the needed base measures (i.e. to create new or reuse existing *monitoring probes*) into the application. For example, user authentication is a security objective that has an authentication-level indicator. The value for that indicator is calculated with an analysis model, whose selection depends on the used security mechanism. Consequently, there are different analysis models for password authentication and fingerprint authentication. Furthermore, these mechanisms apply different base measures.

In heterogeneous smart spaces, *attributes* to measure and applicable *base measures* change due to environment changes. For instance, in some situations a device communicates with a simple sensor device via ZigBee and after a while with a laptop computer via Wi-Fi. The ISMO makes it possible to store an extensive set of base measures and analysis models, which supports monitoring in heterogeneous environments.

All versions of the security ontology utilised in the original publications and cases distinguish measures and security knowledge. The first version, presented in PIV and C1, does not present mappings from measures to security objectives and mechanisms. Therefore, the architect has to select measures without any aid from the ontology. Consequently, the ISMO defines mappings between measures and security knowledge. Hence, the architect can find the appropriate measures from the ISMO by following the *connections* from the security objectives to the mechanisms.

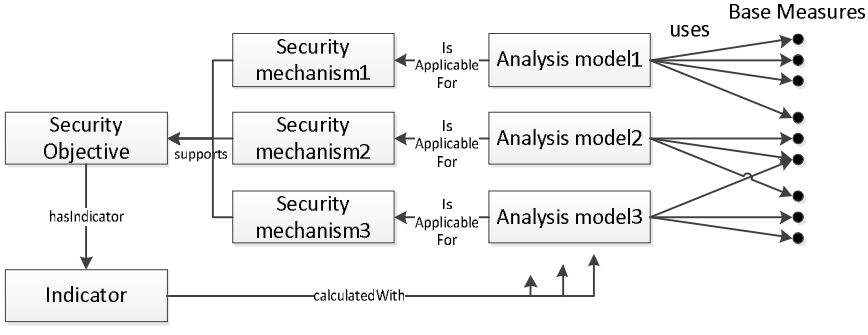


Figure 12. ISMO content for design-time usage.

In addition, the knowledge base research objective states that it must be possible to extend the content of the knowledge base. Possibilities to extend the ISMO support its utilisation in heterogeneous smart spaces. PVI and C4 present how to extend the ISMO. The case shows how to extend the ISMO with *new base measures* by adding the *required attribute* and *measurement method* for the base measure. Hence, future analysis models are able to utilise these base measures in order to take security mechanism reliability into account. Clearly, possibilities to extend the ISMO at design-time are wider than runtime extension possibilities. The architect has to implement the selected base measures into the application, and thus, new base measures can only be added to the ISMO at design-time.

3.3.3 Ontology usage at runtime

The runtime usage is covered by PIII, PV and PVII, and cases C2, C3 and C5. As already said, the ISMO is developed iteratively, and thus, the runtime usage is also developed onwards through publications and cases. Therefore, PVII and C5 present the latest means for the runtime usage. In contrast to design-time usage, the smart space application is an entity that retrieves knowledge from the ISMO at runtime. That is, knowledge has to be in a machine-readable form, which is achieved by utilising an OWL-formatted ontology. Figure 13 and Figure 14 illustrate example situations where implementation components (component symbols in the bottom parts in the figures) utilise the content from the ISMO (the upper parts in the figures). The legend for these figures is defined in Figure 5. Furthermore, dashed arrows indicate the connections between the ISMO and the implementation components.

Figure 6 shows that the Monitor, Analyser and Planner components retrieve knowledge from the ISMO. Firstly, the Monitor component retrieves the base measures that are applicable for the current situation. Thus, monitoring probes are utilised only if needed. For example, it is not reasonable to utilise all the monitoring probes if the adaptation requires only base measures for authentication purposes. Figure 13 contains four base measure instances and the related monitoring probe implementations.

Next, knowledge is retrieved from the ISMO by the Analyser component. The purpose is to find the analysis model that calculates the security-level indicator for the required security objective. The analysis model selection has to take into account the utilised security mechanism. As mentioned in the previous section, separate analysis models are needed when the security objective is fulfilled with different mechanisms. Figure 13 presents a situation where the *Analyser* component uses the *analysis model* for password authentication (i.e. the instantiated analysis model *is applicable for* the *password authentication security mechanism*). The presented analysis model is utilised in C5 and it is extended from the analysis model initially developed in PV and C3. In other words, the utilisation of analysis models supports extensibility. The analysis model in the figure uses the *password type indicator* and the analysis model for it is also defined in PV and C3. Analysis models are a mandatory part to recognise the *achieved security level* and adaptation need. In this dissertation analysis models are developed for the validation cases, and thus, the purpose is not to define the complete set of analysis models. However, the analysis models developed for these cases offer a base to define analysis models for different security objectives and mechanisms in the future.

As visible from Figure 13, analysis models are described by means of English and Boolean algebra. Cases C3 and C5 utilise a component that is able to parse these analysis models and calculate the security-level indicators from the results of base measures. C3 contains a separated parser component called the *Analysis model parser*, while in C5 this is an internal part of the Analyser component as presented in Figure 7.

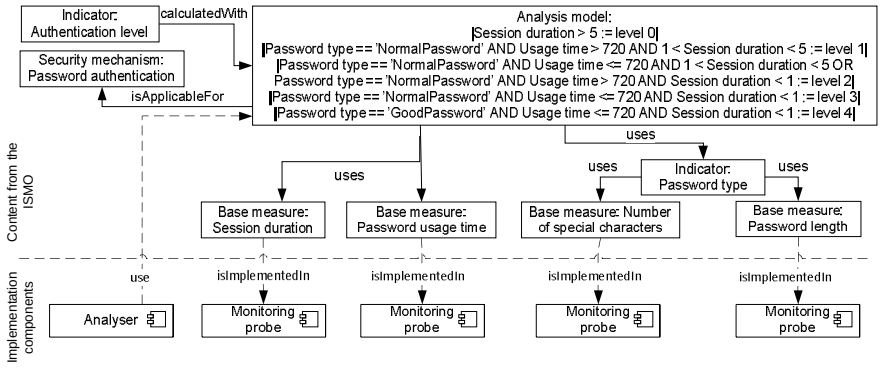


Figure 13. Sample knowledge for the Monitor and Analyse phases.

Lastly, the Planner component retrieves input knowledge from the ISMO in order to create the *adaptation plan*. In this phase, the purpose is to find *security mechanism* alternatives and *attributes* to change (c.f. PIII and PVII). In PIII and C2, security mechanisms that support the required security objectives are retrieved. In contrast, PVII and C5 retrieve both mechanisms and attributes, which can be adapted. Figure 14 shows an example situation where the adaptation plan is created based on the knowledge from the ISMO (c.f. the second alternative in Figure 9). From the ISMO, the Planner component retrieves *Action to adapt* instances for attributes and security mechanisms by means of the *adaptableWith* property. Implementation components contain concrete implementations to perform these adaptations. Consequently, the Planner component gets the knowledge that makes it possible to create an adaptation plan. In other words, knowledge shows the places to adapt (mechanisms and attributes) that are able to affect the achieved security level. The adaptation utilises goal-orientated approach, that is, the adaptation plan that firstly satisfies the requirement is selected.

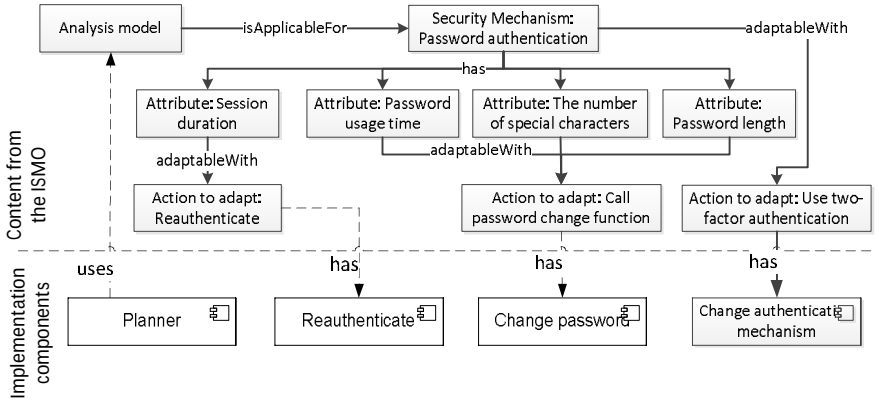


Figure 14. Sample of knowledge utilisation in the Plan phase.

In this dissertation security knowledge is separated from the adaptation loop. Moreover, it is necessary to define knowledge requirements for different adaptation phases. Hence, Table 6 summarises the knowledge required in the adaptation phases.

Table 6. Adaptation phases and required knowledge.

Adaptation phase	Knowledge from the ISMO
Monitoring	Base measures Measurement methods
Analysing	Derived measures Indicators Measurement functions Analysis models
Planning	Action to adapt Attributes Security mechanisms

Extending the content of the knowledge base is also possible at runtime. Naturally, runtime extending possibilities are not as wide as at design-time. Nevertheless, new analysis models can appear and the existing ones can be modified at runtime. Thus, the approach supports a dynamic planning process. This is achieved because knowledge is not hard coded into the adaptation loop or into the application.

4. Discussion

This dissertation concentrates on security adaptation in smart spaces. An architecture that contains the whole adaptation loop has not been presented in any of the existing security-adaptation approaches. In addition, the required knowledge is not tackled in the existing approaches. These issues have motivated this work.

In this section, the research objectives are revisited in Sub-section 4.1, and Sub-section 4.2 discusses contributions. After that, Sub-section 4.3 compares contributions to the existing approaches and Sub-section 4.4 discusses the limitations of the results and possible future work.

4.1 Research objectives revisited

The following research problem was stated in the introduction: How to achieve security adaptation in a smart space application? In order to solve the problem it was further divided into two research objectives. The first research objective was to define the architecture for security adaptation. The architecture has to contain the whole adaptation loop, starting from observations and ending with the execution of the adaptation. In addition, it was required to develop a means of monitoring the achieved security level. The second objective was to develop a knowledge base that is applicable both at design-time and runtime. Hence, knowledge has to be presented in a form which is reachable for the software architect and the adaptive application. Moreover, it was required that both the architecture and the knowledge base support reusability and extensibility.

The architecture objective is achieved as follows. The original publication PI envisions a loop structure for adaptation. The loop structure is a natural selection for adaptation that occurs continuously at runtime – as visible from the surveys [4,5,41]. PII and PIII add context monitoring and ontology utilisation into the adaptation architecture. Use case C2 is the first validation performed for the architecture, which shows that the developed ontology is an applicable form for the knowledge base. Moreover, it indicates that the security-adaptation architecture contains all the necessary phases for adaptation. Nevertheless, some necessary enhancements are also recognised. Firstly, it is mandatory to further develop the monitoring part because the structure of the measures is not formulated exactly. Secondly, the phases of the adaptation architecture are not in balance. In other

words, some phases are small, trivial actions but other phases are complex and not defined in detail. Therefore, the monitoring part is developed further in PV and C3. In that publication, monitoring is divided into generic and implementation-specific parts. As a result, the architecture offers a better solution from the extensibility and reusability points of view. The last version of the security adaptation architecture is developed in PVII and validated in C5 (i.e. SUM-SS pilot). The developed architecture conforms to the MAPE-K reference model and separates knowledge from the adaptation loop to an ontology-based knowledge base. Moreover, the architecture utilises security measures in order to monitor the achieved security level.

The knowledge base objective is achieved as follows. From the design-time viewpoint, the knowledge base is developed in PIV and PVI. The first-mentioned publication utilised quality ontologies, namely the security and reliability ontologies, for modelling quality requirements and transforming requirements into architectural models. In parallel, C1 validates how the software architect utilises ontologies as the knowledge base at design-time. PVI and C4 in turn, show how to extend the knowledge base with new design-time knowledge. From the runtime point of view, the knowledge base is developed in PI, PIII, PV and PVII. The publication PI compares security ontologies from the runtime applicability viewpoint and PIII presents a structural sketch for an ontology-based knowledge base. In C2, the security ontology is utilised for the first time at runtime. Next, the ISMO ontology is described in PV, and thus, the publication makes the main contribution to the knowledge base. In tandem, PV presents use case C3, which validates the ISMO. The latest version of the knowledge base is presented in PVII and validated in C5. The validation proves that an ontology-based knowledge base is applicable during the whole adaptation loop.

4.2 Main contributions

This dissertation makes three research contributions. Firstly, security adaptation architecture that conforms to the MAPE-K reference model and separates security knowledge from the adaptation loop is developed. Secondly, the architecture supports the utilisation of security measures to recognise an adaptation need. Lastly, security ontology is developed and applied for the security adaptation as the knowledge base. Next, these contributions are discussed one by one.

Contributions for architecture: The latest version of the architecture strictly conforms to the MAPE-K reference model. In other words, the contribution covers all the phases of the MAPE loop and describes their intended usage. Hence, the proposed architecture differs from the existing ones, which ignore some phases, or alternatively, combine some adaptation phases. Applying the MAPE-K reference model facilitates the utilisation of the architecture because the phases of the adaptation loop are widely understood. The presented security adaptation architecture is the first one that specifically separates security knowledge from the adaptation loop. The knowledge requirements are taken systematically into ac-

count by utilising ontology (i.e. the ISMO). Moreover, the knowledge required in the adaptation phases is described. The knowledge separation makes it possible that hard-coded analysis models or adaptation rules are not needed. For this reason, modifying and extending the knowledge does not cause a need to modify the application logic. Furthermore, both the structure and behaviour of the architecture are described. In parallel, the architecture supports extensibility and reusability evolution qualities. Consequently, software architects are able to utilise the architecture in their own applications. Finally, the architecture is built from the adaptation viewpoint, instead of solving a particular security-related adaptation problem. Hence, the solution is generic from the security objective, mechanism and asset viewpoints alike, which supports usage in heterogeneous smart spaces.

Contribution to recognise the adaptation need: The presented architecture supports the utilisation of security measures to recognise an adaptation need. First, risk-based measures for the integrity and confidentiality of the communication are applied. Afterwards, security measures are presented by means of a three-level structure (i.e. base measures, derived measures and indicators). This makes it possible to define the measures exactly and to achieve systematic monitoring. Hence, the possibilities of understanding the monitoring functionality and analysing its reliability and feasibility are enhanced from the existing approaches. Moreover, the three-level structure ensures that new measures can be created easily by reusing existing base measures, which is not supported in the existing approaches. The three-level structure makes it possible to divide the monitoring into a generic part and an implementation-specific part. Thus, new derived measures and indicators can be provided for an application, even at runtime. This is an important feature in changing smart spaces where different situations cause variations. Furthermore, each base measure creates a straight connection to a concrete monitoring probe by means of a measurement method. Finally, the dissertation shows that security measure-based monitoring is able to act as a part of the security-adaptation architecture, and security measures are able to offer valuable information for the planning phase of the adaptation loop. The utilised security measures are related to password-based user authentication. However, the presented three-level measurement structure and monitoring by means of generic and implementation specific parts are not dependant on the utilised measures. Consequently, the same structure is applicable for measures intended for other security objectives.

Contribution to the knowledge base: In the dissertation, the ontology is first time applied to the security adaptation as the knowledge base. The dissertation defines the structure and content of the ontology-based knowledge base, which are not defined in the previous security adaptation approaches. The knowledge base ensures that the security and measuring knowledge is represented in a unified manner and is accessible for both software architects and applications. The most significant contribution for the knowledge base is the ISMO that merges existing security and measurement ontologies, and thus, the ISMO is an extensive ontology, which is compatible with its predecessors. Reusing the existing ontologies was a reasonable selection. Firstly, the reuse ensured a much wider ontology

without overlapping terms. Secondly, the content and structure of the reused ontologies had already been validated and accepted in previous scientific publications, which brings a higher-maturity level. Lastly, reusing is one preferred ontology-development method suggested in [84]. Knowledge requirements differ for design-time and runtime usage, and thus, it was required that the knowledge base is applicable at design-time and runtime alike. From the design-time viewpoint, the ISMO facilitates a software architect to select the applicable security mechanisms for security objectives. Moreover, the ISMO offers the security measures for the software architect. From the runtime viewpoint, knowledge from the ISMO is utilised in the monitoring, analysing and planning phases. It is shown in use cases that the ontology is an appropriate knowledge form even for runtime usage in mobile devices. Reusability was one requirement for the knowledge base. The knowledge base is utilised in three runtime and two design-time use cases in order to achieve evidence about the reusability and extensibility possibilities of the knowledge base. Use cases show that the knowledge base can be reused in various situations. In addition, the extension possibilities of the knowledge base are described by adding new design-time-related measures into the ISMO.

4.3 Comparison to the related work

This section compares the presented contributions to the related work. Firstly, the main differences are described in a textual form. Thereafter, the benchmarking criteria defined in Sub-section 1.5 is utilised in **Table 7** to compare contributions to the related work in more formal way. The similar comparison in the more extensive form is presented in our recent publication in [85].

In **Table 7** a line (-) symbol indicates that the aspect is not described in the approach. *Extensibility* and *Reusability* criteria utilise the following grades: “Yes”, “Partially” and “No”. For Extensibility: “Yes” means that the approach can be extended i) by adding adaptation support for new security objectives and ii) by using new adaptation techniques (i.e. new monitoring, analysing, and planning mechanisms). The “Partially” grade means that the approach supports either security- or adaptation-related extensions. Finally, the “No” grade means that extensions cannot be made, or that they are laborious. In parallel, for Reuse: “Yes” means that both the approach as a whole and its individual parts can be reused. The “Partially” grade indicates that individual parts of the approach can be reused. Lastly, the “No” grade means that reusing is not possible or that it is laborious, for example, the approach is closely related to the particular case or environment.

The existing security adaptation approaches do not cover the whole adaptation loop. Yuan and Malek notice in their study that in the existing approaches the most emphasis is put on the Monitor and Analyse phases and details on how the planning occurs are not given [13]. A similar shortage is visible in the approaches described in Sub-section 2.2.1. Naturally, it is possible to find significant contributions from the existing approaches. However, those are mainly intended for a particular phase of the adaptation or for dedicated security objectives or mecha-

nisms, instead of security adaptation as a whole. For instance, the approach from Hulsebosch et al. [14] is intended for adaptive authentication and access control. Consequently, the approach is not generic from the security-objective viewpoint. The approach in [15] combines the Analyse and Plan phases, which is indicated with parentheses in the *adaptation loop* criterion in **Table 7**. However, from the architecture viewpoint it is reasonable to separate the different phases from each other. The GEMOM approach [61] contains elements for the planning purposes, but the functionality is not described. However, in order to achieve security adaptation, it is mandatory to cover the whole adaptation loop. The lifecycle viewpoint (i.e. the software development aspect) is not present in the previous approaches. This is visible in the *Design / Runtime* criteria in Table 7. The security-adaptation approach presented in this dissertation also pays attention to the software design-time.

Measure-based monitoring offers considerable advantages when compared to the monitoring of existing security-adaptation approaches. Due to explicitly defined measures, monitoring is systematic. In the existing approaches, various aspects are monitored. However, details on performing monitoring are not described. One monitoring alternative is to define and monitor security-related events. An adaptation action is triggered when an event occurs and the defined conditions are true. This ECA-based adaptation is applied, for instance, by Russello et al. [57]. From the existing approaches, only GEMOM [61] utilises security measures to trigger adaptation. In GEMOM, measures are produced by means of the decomposition approach [46,47], and thus, the measures form a similar kind of hierarchical structure as described in this dissertation. However, this dissertation maps base measures and their monitoring probes to the Monitoring phase, whereas higher-level measures (derived measures and indicators) are intended for analysing purposes. Consequently, security measuring is not one single component. The three-level measuring structure supports reusability and extensibility qualities, which was one requirement set for the research. This ensures that the Analysing phase can be updated without affecting the monitoring probes. The existing solutions do not concentrate on these evolution qualities.

In the existing security adaptation approaches the knowledge part is not taken into account at a sufficient level. Hulsebosch et al. [14] do not concentrate on knowledge at all. In contrast, Saxena et al. [15] and the GEMOM approach [61] define a database (DB) for input knowledge in their approaches. However, the content of a database or its utilisation are not defined. Even the MAPE-K model [3], does not define the form or content of knowledge. Furthermore, the survey from Huebscher et al. [5] denotes that the division between planning and knowledge in the MAPE-K model is not distinct. The knowledge base presented in this dissertation tackles these challenges. Firstly, the security-adaptation architecture separates the knowledge base from the adaptation loop. Secondly, the form and content of the knowledge base is also defined. Lastly, the usage of knowledge in the different phases of the adaptation loop is described.

Table 7. Benchmarking contributions to the existing approaches.

Criterion	This work	Hulsebosch [14]	Rusello [57]	Saxena [15]	GEMOM [61]
Adaptation loop	MAPE	MA	(MA)PE	M(AP)	MA(PE)
Structure / Behaviour	Both	Both	Structure	Both	Structure
Supported security adaptation	Generic	Authentication, access control	Generic	Generic	Generic
Adaptation need recognition	Security measuring	Applies user's location.	Context monitoring services.	Beforehand selected events	Security measuring
Extensibility	Yes	No	Yes	Yes	Partially
Reusability	Yes	Partially	Yes	Yes	Partially
Design / Runtime	Both	Runtime	Runtime	Runtime	Runtime
Knowledge storage	Separated ontology	-	Integrated	Profile DB	DB
Knowledge form	OWL	-	ECSA policies	-	-
Knowledge content	Security & measuring	-	-	-	-
Knowledge usage	Defined	-	-	-	-

4.4 Limitations and future work

Based on the use cases, the presented architecture is a reasonable solution for security adaptation in smart spaces. The architecture describes the required elements and their mutual behaviour. Nevertheless, the internal content of these elements has to be developed further. The Monitoring phase is defined on a detailed level but the Analyse and Plan phases need refinements. The Analyse phase has to recognise the achieved security level based on the monitoring results and to deduct the required security level from context information. Both of these are complex tasks, which require a lot of knowledge. Developing these further would ensure that the smart space application is able to recognise security requirements and adaptation needs in various situations. Defining security requirements at design-time is extensively covered in [86]. Moreover, Salehie et al. concentrate on the variability of assets and how this affects security in [87]. Hence, one alternative is to apply these results to recognise security requirements in the future. Moreover, the Plan phase of the adaptation architecture will need more sophisticated decision-making algorithms. This dissertation contains few alternatives on how the adaptation plan can be created. However, these are not

sufficient mechanisms as such, for instance in complex situations when trade-offs or mechanism matchmaking between devices have to be taken into account.

Trade-offs have to be considered between security objectives, and in situations where achieving the required security level would cause a failure to meet other quality requirements. For instance, security mechanisms might consume too much computation and network resources, or reduce usability. Consequently, trade-offs between security and other quality requirements have to be analysed. Possible alternative is to apply utility functions as presented in [88] for security and performance trade-offs. On the other hand, it is possible that devices in the smart space produce conflicting adaptation plans. For example, smart space infrastructure creates the adaptation plan to utilise biometric authentication while user's device creates the adaptation plan where two-factor authentication is preferred. In order to solve this mismatch, devices have to be able to negotiate and matchmake. Therefore in the future, both trade-offs and matchmaking have to be taken into account. Currently, the Plan phase seems to be a reasonable phase for these functionalities. Adding trade-offs and matchmaking requires that the internal structure and behaviour of the Planner component is designed carefully in order to avoid bottlenecks and to maintain reusability and extensibility. In addition, it would be reasonable to search existing trade-off and matchmaking approaches and apply the available solutions instead of starting from scratch. Evidently, these functionalities will also need knowledge. Adding trade-off and matchmaking-related knowledge into the ISMO ensures that the hard-coded knowledge in the Plan phase can be minimised even after the inclusion of trade-offs and matchmaking functionalities.

During the research, security measures for confidentiality, integrity and user authentication are utilised. C2 applies measures for confidentiality and integrity, whereas, authentication measures are utilised in C3 and C5. Therefore, only user-authentication measures are presented by utilising a three-level measurement structure and applied in the latest version of the adaptation architecture. However, reliable calculation of the security level requires that a wide set of correctly defined analysis models are available. Thus, it is necessary to define the analysis models for other security objectives in the future. It is notable that each security objective needs separate analysis models for each security mechanism, as visible in Figure 12. Currently, the amount of existing analysis models restricts the direct utilisation of the presented approach. However, the original publications present analysis models to calculate the achieved authentication level, and those models can be applied as a guideline when defining new models in the future. Defining analysis models is a complex and time-consuming task. Therefore, in the future a tool support for analysis model definition is needed. Furthermore, it will be fruitful to evaluate the reliability and feasibility of the currently used authentication measures. These measures were developed based on the available information by using the bottom-up measurement development approach [48]. However, this approach may lead to a situation in which some important factors are not taken into account because the related information is not available.

The presented knowledge base, that is, the ISMO, is the composition of two previously presented ontologies. Hence, its security and measurement parts were validated and accepted in previous scientific publications. These two ontologies are combined with new properties defined in PV. However, it was noticed that these mapping properties are dependent on security measures. Thus, additional mappings have to be defined in the future when new security measures are added. Naturally, new security knowledge will be generated by the research community, and thus, updating and enhancing the knowledge base in the future is necessary. As a result of this, it would be appropriate to establish a community to maintain the knowledge base. In this way, the knowledge base can contain fresh knowledge and support adaptation in a reasonable way. The community could also offer various analysis models and reusable monitoring probes, which will support the evolution of analysis models.

The contributions of this dissertation are intended to offer adaptive security for smart space applications. However, these solutions are also able to damage the existing security. For example, the Monitoring phase may collect information, which can be exploited during an attack. Or alternatively, an attacker may launch an attack towards the Planning phase, in order to cause an adaptation that deactivates the particular security mechanism. These are example cases where the presented security-adaptation approach creates new threats and attack possibilities. Naturally, these kinds of situations are not acceptable. Thus, the presented adaptation approach has to be implemented in such a way that valuable data and adaptation actions are protected. At least the following is required: the confidentiality and integrity of results from monitoring probes; integrity for the Analyse and Plan phases and adaptation plans; and integrity of the ISMO. Moreover, appropriate access control rules are required for the elements in the adaptation architecture. For example, only the Executor component is able to call the Action to Adapt components, and modifying the ISMO requires more permissions than reading. These security aspects have to be studied in the context of future research.

In the research scoping, smart spaces was set as a domain for the research results. However, other domains are also able to gain benefits from the adaptive security, but currently we don't have evaluation use cases that provide evidence for that. Consequently, the next step would be to define a use case outside the smart space environment. For instance, one alternative is to utilise the approach as part of a web browser to adapt the utilised security mechanisms based on content and criticality of information.

The current deployment centralises the adaptation loop and the ISMO in a device where the adaptation is intended to occur, as described in the Deployment Sub-section 3.2.3. The performed use cases indicate that this is a reasonable selection. However, certain cases in the future might need decentralised deployment. Firstly, the resource restrictions of the device under the adaptation might cause a need for decentralisation. For instance, the storage capacity might not be sufficient for the ISMO, or alternatively, the computation resources might not be able to produce the adaptation plan. This is a relevant case especially in embedded devices, which are dedicated to particular usage. Secondly, trade-off analysis

and matchmaking might increase the amount of required communication in the future. Thus, it might be reasonable to perform the Plan phase remotely in one place within the smart space. Consequently, unnecessary communication due to planning can be avoided and only a ready-made adaptation plan is delivered for the adapted device. However, in this case the trustworthiness of the planning device has to be evaluated extensively.

Currently, the ISMO is located within the device where the adaptation occurs. It would be reasonable to consider this deployment issue in the future from the updating viewpoint. In this deployment, updates for the ISMO mean that each device has to update its own instance from the ISMO. Thus, locating the ISMO in one place – accessible via a smart space infrastructure – would offer the most up-to-date knowledge for each device. This can be compared with the current antivirus software, where virus signatures are retrieved from a trusted cloud service.

5. Conclusions

Smart spaces offer enriched services and information for end users. Smart spaces are heterogeneous and dynamic: they contain devices from different vendors and new services and devices may appear constantly. From the security point of view, dynamically changing smart spaces are a challenge, as static and beforehand selected security mechanisms are not able to offer an optimal security level for the varying situations. Moreover, it is impossible at design-time to anticipate all situations in which a smart space application will be utilised. These challenges cause a need for self-adaptive security, which is able to select security mechanisms and tune their parameters at runtime.

The research problem of this dissertation was to achieve security adaptation in a smart space application. The research problem was further divided into architecture and knowledge base objectives. The purpose of the architecture objective was to develop a security adaptation architecture that covers the whole adaptation loop. In addition, it was required to develop a means to monitor the achieved security level in order to recognise the adaptation need. In contrast, the knowledge base objective required the development of a separate knowledge base for security. It was required that the knowledge base offer adaptation knowledge for design-time and runtime purposes alike.

Both architecture and knowledge base research objectives were developed iteratively through the dissertation work. The original publications PI–PIII, PV and PVII and use cases C2, C3 and C5 contributed to the architecture objective. The knowledge base was developed in all use cases and original publications. Due to the iterative development, the most recent publication PVII and use case C5 contain the latest version of the presented security-adaptation approach.

As the final conclusion, the contributions of the dissertation are: Firstly, a reusable adaptation architecture for security is presented. The architecture strictly conforms to the MAPE-K model and defines all phases in it. The proposed architecture is the first solution that specifically separates security knowledge and the adaptation loop. Secondly, the architecture supports the utilisation of security measures in order to recognise the adaptation need. Security measures are presented by means of a three-level structure, which supports systematic monitoring. Due to the suggested architecture, it is possible to reuse and extend the defined security measures. Thirdly, the proposed solution is the first time that an ontology has been applied for security adaptation. The Information Security Measuring

Ontology (ISMO) acts as the knowledge base for the adaptation. Moreover, the ISMO is applicable at design-time and runtime alike. These contributions form the first approach to combine the MAPE-K reference model, security measuring and knowledge from ontologies to form a consistent security adaptation architecture, which is able to offer security adaptation for smart space applications – as required by the research problem.

References

1. Weiser, M. The Computer for the 21st Century. Scientific American **1991,** 265, 94–104.

2. Conti, M.; Das, S. K.; Bisdikian, C.; Kumar, M.; Ni, L. M.; Passarella, A.; Roussos, G.; Tröster, G.; Tsudik, G.; Zambonelli, F. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence. Pervasive Mob. Comput. **2012,** 8, 2–21.

3. Kephart, J. O.; Chess, D. M. The vision of autonomic computing. Computer **2003,** 36, 41–50.

4. Salehie, M.; Tahvildari, L. Self-adaptive software: Landscape and research challenges. ACM Trans.Auton.Adapt.Syst. **2009,** 4, 14:1–14:42.

5. Huebscher, M. C.; McCann, J. A. A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **2008,** 40, 7:1–7:28.

6. ISO/IEC 9126-1:2001 Software Engineering – Product Quality – Part 1: Quality Model, International Organization of Standardization 2001.

7. Avižienis, A.; Laprie, J.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **2004,** 1, 11–33.

8. ISO/IEC 15408-1:2009 Common Criteria for Information Technology Security Evaluation – Part 1: Introduction and general model, International Organization of Standardization 2009.

9. Stallings, W. Cryptography and Network Security – Principles and Practice, 5th ed., Prentice Hall, New Jersey, 2010.

10. Stoneburner, G.; Goguen, A.; Feringa, A. Risk management guide for information technology systems. Special publication 800-30 **2002,** 1–41.

11. Weyns, D.; Iftikhar, M. U.; Malek, S.; Andersson, J. Claims and supporting evidence for self-adaptive systems: A literature study. In Proceedings of the Software Engineering for Adaptive and Self-Managing Systems Workshop on ICSE, Zurich, Switzerland, 4–5 June, IEEE, 2012, pp. 89–98.

12. Elkhodary, A.; Whittle, J. A Survey of Approaches to Adaptive Application Security. In Proceedings of the Software Engineering for Adaptive and

Self-Managing Systems Workshop, Minneapolis, 20–26 May, IEEE, 2007, pp. 16–23.

13. Yuan, E.; Malek, S. A taxonomy and survey of self-protecting software systems. In Proceedings of the Software Engineering for Adaptive and Self-Managing Systems, Zürich, Switzerland, 4–5 June, IEEE, 2012, pp. 109–118.

14. Hulsebosch, R.; Bargh, M.; Lenzini, G.; Ebben, P.; Iacob, S. Context sensitive adaptive authentication. In Smart Sensing and Context, Kortuem G., Finney J., Lea R., Sundramoorthy V., Eds.; Springer Berlin Heidelberg, 2007, pp. 93–109.

15. Saxena, A.; Lacoste, M.; Jarboui, T.; Lücking, U.; Steinke, B. A Software Framework for Autonomic Security in Pervasive Environments. In Information Systems Security, McDaniel P., Gupta S., Eds.; Springer Berlin / Heidelberg, 2007, pp. 91–109.

16. Järvinen, P. On Research Methods, Opinpajan kirja, Tampereen yliopistopaino Oy, Tampere, Finland, 2004.

17. Pantsar-Syväniemi, S. Reusable, semantic, and context-aware micro-architecture. Approach to managing interoperability and dynamics in smart spaces, VTT, Espoo, Finland, 2013.

18. Savolainen, P.; Niemelä, E.; Savola, R. A taxonomy of information security for service centric systems. In Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, Lübeck, Germany, 27–31 August, IEEE, 2007, pp. 5–12.

19. Zhou, J.; Niemela, E.; Evesti, A.; Immonen, A.; Savolainen, P. OntoArch Approach for Reliability-Aware Software Architecture Development. In Proceedings of the 32nd International conference on Computer Software and Applications, Turku, Finland, 28 July – 1 August, IEEE, 2008, pp. 1228–1233.

20. García, F.; Bertoa, M. F.; Calero, C.; Vallecillo, A.; Ruíz, F.; Piattini, M.; Genero, M. Towards a consistent terminology for software measurement. Inf. and Softw. Technol. **2006,** 48, 631–644.

21. Herzog, A.; Shahmehri, N.; Duma, C. An ontology of information security. Journal of Information Security and Privacy **2007,** 1, 1–23.

22. Cook, D.; Das, S. Smart environments: Technology, protocols and applications, Wiley-Interscience, New Jersey, 2004.

23. Cook, D.; Das, S. K. How smart are our environments? An updated look at the state of the art. Pervasive Mob. Comput. **2007,** 3, 53–73.

24. Ovaska, E.; Salmon Cinotti, T.; Toninelli, A. Design principles and practices of interoperable smart spaces. In Advanced Design Approaches to Emerging Software Systems: Principles, Methodologies, and Tools, Liu X., Li Y., Eds.; IGI Global, 2011, pp. 18–47.

25. Pantsar-Syväniemi, S.; Purhonen, A.; Ovaska, E.; Kuusijärvi, J.; Evesti, A. Situation-Based and Self-Adaptive Applications for Smart Environment. J. Ambient Intelligence and Smart Environ. **2012,** 4, 491–516.

26. Chen, G.; Kotz, D. A Survey of Context-Aware Mobile Computing Research. **2000,** Technical Report TR2000-381.

27. Raychoudhury, V.; Cao, J.; Kumar, M.; Zhang, D. Middleware for pervasive computing: A survey. Pervasive Mob. Comput. **2013,** 9, 177–200.

28. Honkola, J.; Laine, H.; Brown, R.; Tyrkkö, O. Smart-M3 information sharing platform. In Proceedings of the Symposion on Computers and Communications, Riccione, Italy, 22–25 June, IEEE, 2010, pp. 1041–1046.

29. Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. Pattern-oriented software architecture – A system of patterns, John Wiley, Chichester, England, 1996.

30. RDF Primer. http://www.w3.org/TR/rdf-primer/, 2004, Accessed 23 November 2012.

31. Smart-M3. http://sourceforge.net/projects/smart-m3/, 2012, Accessed 23rd November 2012.

32. Evesti, A.; Eteläperä, M.; Kiljander, J.; Kuusijärvi, J.; Purhonen, A.; Stenudd, S. Semantic Information Interoperability in Smart Spaces. In Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia, Cambride, UK, 22–25 November, ACM, 2009, pp. 158–159.

33. Suomalainen, J.; Hyttinen, P.; Tarvainen, P. Secure information sharing between heterogeneous embedded devices. In Proceedings of the 4th European Conference on Software Architecture: Companion Volume, Copenhagen, Denmark, 23–26 August, ACM, 2010, pp. 205–212.

34. Dierks, T. and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. http://www.ietf.org/rfc/rfc5246.txt, 2008, Accessed 23 Nov 2012.

35. Bishop, M. Computer Security – Art and Science, Addison-Wesley 2003.

36. Anderson, R. J. Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed., Wiley, Indianapolis, 2008.

37. Herrmann, D. S. Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI, Auerbach Publications, Boca Raton, 2007.

38. Jansen, W. Directions in Security Metric Research. NISTIR 7564 **2009,** 21.

39. Andersson, J.; Lemos, R.; Malek, S.; Weyns, D. Modeling Dimensions of Self-Adaptive Software Systems. In Software Engineering for Self-Adaptive Systems, Cheng B. C., Lemos R., Giese H., Inverardi P., Magee J., Eds.; Springer, Berlin Heidelberg, 2009, pp. 27–47.

40. Dobson, S.; Denazis, S.; Fernández, A.; Gaïti, D.; Gelenbe, E.; Massacci, F.; Nixon, P.; Saffre, F.; Schmidt, N.; Zambonelli, F. A survey of autonomic communications. ACM Trans.Auton.Adapt.Syst. **2006,** 1, 223–259.

41. Psaier, H.; Dustdar, S. A survey on self-healing systems: approaches and systems. Computing **2011,** 91, 43–73.

42. Parashar, M.; Hariri, S. Autonomic Computing: An Overview. In Unconventional Programming Paradigms, Banâtre J., Fradet P., Giavitto J., Michel O., Eds.; Springer, Berlin Heidelberg, 2005, pp. 257–269.

43. Kephart, J. O.; Walsh, W. E. An artificial intelligence perspective on autonomic computing policies. In Proceedings of the 5th Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, USA, 7–9 June 2004, IEEE, 2004, pp. 3–12.

44. ISO/IEC 14598-1 Information Technology – Software Product Evaluation Part 1 General Overview, 1999.

45. Savola, R. M.; Abie, H. On-Line and off-line security measurement framework for mobile ad hoc networks. Journal of Networks **2009,** 4565–579.

46. Wang, C.; Wulf, W. A. Towards a Framework for Security Measurement. In Proceedings of the 20th National Information Systems Security Conference, Baltimore, Maryland, October, 1997, pp. 522–533.

47. Savola, R.; Abie, H. Development of measurable security for a distributed messaging system. Int. J. on Advances in Security **2009,** 2, 358–380.

48. Payne, S. C. A Guide to Security Metrics. SANS InfoSec Reading Room **2007,** 1–11.

49. Svahnberg, M.; Van Gurp, J.; Bosch, J. A taxonomy of variability realization techniques. Software: Practice and Experience **2005,** 35, 705–754.

50. Bosch, J. Design & Use of Software Architectures: Adopting and evolving a product-line approach, Addison-Wesley 2000.

51. Bass, L.; Clements, P.; Kazman, R. Software architecture in practice, 2nd ed., Addison-Wesley, Boston, 2003.

52. Niemelä, E.; Evesti, A.; Savolainen, P. Modeling quality attribute variability. In Proceedings of the 3rd International Conference on Evaluation of Novel Approaches to Software Engineering, Funchal, Madeira, 4–7 May, 2008, pp. 169–176.

53. Hashii, B.; Malabarba, S.; Pandey, R.; Bishop, M. Supporting reconfigurable security policies for mobile programs. Computer Networks **2000,** 33, 77–93.

54. Hu, W.; Hiser, J.; Williams, D.; Filipi, A.; Davidson, J. W.; Evans, D.; Knight, J. C.; Nguyen-Tuong, A.; Rowanhill, J. Secure and practical defense against code-injection attacks using software dynamic translation. In Proceedings of the 2nd international conference on Virtual execution environments, Ottawa, Canada, 14–16 June, ACM, 2006, pp. 2–12.

55. Knight, J.; Strunk, E. Achieving Critical System Survivability Through Software Architectures. In Architecting Dependable Systems II, de Lemos R., Gacek C., Romanovsky A., Eds.; Springer Berlin / Heidelberg, 2004, pp. 69–91.

56. Ryutov, T.; Zhou, L.; Neuman, C.; Leithead, T.; Seamons, K. E. Adaptive trust negotiation and access control. In Proceedings of the 10th ACM symposium on Access control models and technologies, Stockholm, Sweden, 1–3 June, ACM, 2005, pp. 139–146.

57. Russello, G.; Dulay, N. An Architectural Approach for Self-Managing Security Services. In Proceedings of the International Conference on Advanced Information Networking and Applications Workshops, Bradford, Yorkshire, 26–29 May, IEEE, 2009, pp. 153–158.

58. Russello, G.; Mostarda, L.; Dulay, N. ESCAPE: A Component-Based Policy Framework for Sense and React Applications. In Component-Based Software Engineering, Chaudron M., Szyperski C., Reussner R., Eds.; Springer Berlin / Heidelberg, 2008, pp. 212–229.

59. Russello, G.; Mostarda, L.; Dulay, N. A policy-based publish/subscribe middleware for sense-and-react applications. J. Syst. Software **2011,** 84, 638–654.

60. Hulsebosch, R. J.; Salden, A. H.; Bargh, M. S.; Ebben, P. W. G.; Reitsma, J. Context sensitive access control. In Proceedings of the 10th symposium on Access control models and technologies, Stockholm, Sweden, 1–3 June, ACM, 2005, pp. 111–119.

61. Abie, H.; Savola, R. M.; Bigham, J.; Dattani, I.; Rotondi, D.; Da Bormida, G. Self-Healing and Secure Adaptive Messaging Middleware for Business-Critical Systems. Int. J. on Advances in Security **2010,** 3, 34–51.

62. Abie, H. Adaptive security and trust management for autonomic message-oriented middleware. In Proceedings of the 6th International Conference on Mobile Adhoc and Sensor Systems, Macau, China, 12–15 October, IEEE, 2009, pp. 810–817.

63. Garlan, D.; Shang-Wen, C.; An-Cheng, H.; Schmerl, B.; Steenkiste, P. Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **2004,** 37, 46–54.

64. Menasce, D.; Gomaa, H.; Malek, S.; Sousa, J. P. SASSY: A Framework for Self-Architecting Service-Oriented Systems. IEEE Software 2011, 28, 78–85.

65. Gruber, T. R. A translation approach to portable ontology specifications. Knowledge acquisition **1993,** 5, 199–220.

66. Zhou, J. Knowledge Dichotomy and Semantic Knowledge Management. In Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web, Jyväskylä, Finland, 25–27 August, Springer US, 2005, pp. 305–316.

67. Chandrasekaran, B.; Josephson, J. R.; Benjamins, V. R. What are ontologies, and why do we need them? Intelligent Systems and their Applications, IEEE **1999,** 14, 20–26.

68. Denker, G.; Kagal, L.; Finin, T. Security in the Semantic Web using OWL. Information Security Technical Report **2005,** 10, 51–58.

69. Denker, G.; Kagal, L.; Finin, T.; Paolucci, M.; Sycara, K. Security for DAML Web Services: Annotation and Matchmaking. In The Semantic Web – ISWC 2003, Fensel D., Sycara K., Mylopoulos J., Eds.; Springer Berlin Heidelberg, 2003, pp. 335–350.

70. Kim, A.; Luo, J.; Kang, M. Security Ontology for annotating resources. In Proceedings of the On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Agia Napa, Cyprus, 31 October – 4 November, Springer-Verlag Berlin Heidelberg, 2005, pp. 1483–1499.

71. Tsoumas, B.; Gritzalis, D. Towards an Ontology-based Security Management. In Proceedings of the 20th Advanced Information Networking and Applications, Vienna Austria, 18–20 April, IEEE, 2006, pp. 985–992.

72. Blanco, C.; Lasheras, J.; Valencia-García, R.; Fernández-Medina, E.; Toval, A.; Piattini, M. A systematic review and comparison of security ontologies. In Proceedings of the 3rd International Conference on Availability, Security, and Reliability, Barcelona, Spain, 4–7 March, IEEE, 2008, pp. 813–820.

73. Blanco, C.; Lasheras, J.; Fernández-Medina, E.; Valencia-García, R.; Toval, A. Basis for an integrated security ontology according to a systematic review of existing proposals. Computer Standards & Interfaces **2011,** 33, 372–388.

74. Fenz, S.; Ekelhart, A. Formalizing information security knowledge. In Proceedings of the Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, Sydney, Australia, 10–12 March, ACM, 2009, pp. 183–194.

75. Ekelhart, A.; Fenz, S.; Klemen, M.; Weippl, E. Security Ontology: Simulating Threats to Corporate Assets. In Information Systems Security, Bagchi A., Atluri V., Eds.; Springer, Berlin Heidelberg, 2006, pp. 249–259.

76. W3C. Web Ontology Language (OWL). http://www.w3.org/2004/OWL/, 2007, Accessed Jan 25 2013.

77. W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2012, Accessed Jan 25 2013.

78. W3C. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004, Accessed Jan 25 2013.

79. Runeson, P.; Höst, M. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **2009,** 14, 131–164.

80. Palviainen, M.; Evesti, A.; Ovaska, E. The reliability estimation, prediction and measuring of component-based software. J.Syst.Software **2011,** 84, 1054–1070.

81. Sofia Pilot Brochure. http://www.slideshare.net/sofiaproject/sofia-project-brochure-pilots-set, 2012, Accessed 23 November 2012.

82. Protégé. Protégé Ontology tool. http://protege.stanford.edu/, 2006.

83. Pantsar-Syväniemi, S.; Kuusijärvi, J.; Ovaska, E. Supporting Situation-awareness in Smart Spaces. In Proceedings of the International Workshops, S3E, HWTS, Doctoral Colloquium, Held in Conjunction with GPC 2011, Oulu, Finland, 11–13 May 2011, Springer-Verlag Berlin Heidelberg, 2012, pp. 14–23.

84. Noy, N. F.; McGuinness, D. L. Ontology development 101: A guide to creating your first ontology. **2001,** 1–25.

85. Evesti, A.; Ovaska, E. Comparison of Adaptive Information Security Approaches. ISRN Artificial Intelligence **2013,** 1–18.

86. Haley, C. B.; Laney, R.; Moffett, J. D.; Nuseibeh, B. Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Transactions on Software Engineering **2008,** 34, 133–153.

87. Salehie, M.; Pasquale, L.; Omoronyia, I.; Ali, R.; Nuseibeh, B. Requirements-driven adaptive security: Protecting variable assets at runtime. In Proceedings of the 20th International Requirements Engineering Conference (RE), Chicago, USA, 24th–28th September, IEEE, 2012, pp. 111–120.

88. Menascé, D. A.; Ewing, J. M.; Gomaa, H.; Malek, S.; Sousa, J. P. A framework for utility-based service oriented design in SASSY. In Proceedings of the 1st joint WOSP/SIPEW international conference on Performance engineering, San Jose, California, USA, 28–30 January, ACM, 2010, pp. 27–36.

PUBLICATION I

From security modelling to run-time security monitoring

In: Proceedings of the European Workshop on Security in Model Driven Architecture (SECMDA), Enschede, the Netherlands, 24 June 2009. Pp. 33–41.

From Security Modelling to Run-time Security Monitoring

Antti Evesti, Eila Ovaska and Reijo Savola

VTT Technical Research Centre of Finland, Kaitoväylä 1,
90571 Oulu, Finland
{Antti.Evesti, Eila.Ovaska, Reijo.Savola}@vtt.fi

Abstract. In this paper we take the first steps from security modelling to run-time security monitoring. Providing full support for run-time security monitoring requires that following issues are solved: security concepts has to be defined in an unambiguous way, security level has to be defined and measured, and finally, software has to adapt itself based on measurements and requirements. This paper addresses the unambiguous definition of security by examining existing security ontologies. None of the existing ontologies is able to support run-time security monitoring as such, and there is a need to combine and widen these ontologies. In addition, this paper describes our vision how run-time security management can be achieved as the wholeness.

Keywords: Security ontology, security measuring

1 Introduction

Today's software products are not running in a static and beforehand known environment. Instead, software is running in mobile devices within constantly evolving environments or, alternatively, software is running in a fixed place but new services become available for exploitation of this device. In addition, the threat landscape is changing constantly. In both cases, the user wants to preserve a particular security level (or security performance) – even though his/her environment changes. In these circumstances, making all security decisions at a design time is not sufficient and thus managing security also at run-time is required. In other words, it is necessary to reveal security changes, by means of monitoring, and adapt the software during its execution – in order to ensure the desired security level for system's users.

Although there are a number of security solutions introduced in the literature [10], [22], there is no common understanding how to define and measure security in practical cases. The existing definitions are generic but too abstract. One example of immaturity of methods and techniques used for security modelling and measurement is the diversity of security concepts and metrics defined in research papers and standards, for example in [1] and [3]. Moreover, security is a cross-cutting issue [2], and therefore, its management is difficult, not only at run-time but also at design-time.

Development of appropriate run-time security solutions is a complex process. Firstly, security has to be defined, i.e. modelled, in an unambiguous way that is

universally understood by all stakeholders. These generic security models, typically represented by means of security ontologies, are used as basis of security aware software development. Secondly, an appropriate security level needs to be defined, measured and monitored constantly. Obviously, a widely-accepted measurement ontology is needed to enable metrics development. Thirdly, software has to be able to adapt itself based on the measurement results. In this paper, we concentrate on the first challenge, security modelling, by examining security ontologies that can be used for representing security. Since these ontologies offer support for security measurement, security measurement issues are also discussed. Finally, we introduce our vision about the run-time security management based on existing security ontologies that are to be combined and enhanced.

The remainder of this paper is organized as follows. Firstly, existing security ontologies are examined, and thereafter, quality and security measurement issues are discussed. Section 3 shows how our approach takes security issues into account at design-time, and, thereafter, we present our vision of the run-time security management. Conclusions and future work section closes the paper.

2 Security Ontologies

2.1 Information Security in General

In order to achieve coherent understanding of security we have to describe security issues in a universal way and incorporate enough details to make the description useful. Ontologies make it possible to give this kind of presentation. We used the following method to select ontologies to this study: 1) ontologies have to concentrate information security, 2) only general purpose ontologies, i.e. not domain specific ontologies, and 3) ontologies have to be mature enough, i.e. at least a concept structure has to be available. Based on these criteria, we describe four existing security ontologies and consider their differences – concentrating applicability to run-time usage and security measurement.

Savolainen et al. present a taxonomy of information security for service centric systems in [7]. The presented taxonomy is intended for the use of software architects of service centric systems. Thus, the taxonomy supports following aspects: 1) stakeholders' participation in the development of service-centric systems, 2) improve communication of security concerns in requirements elicitation, 3) aid to designing and constructing of security means while architecting and 4) support quality analysis phase by providing a common security terminology. [7]

The security taxonomy is divided to five main concepts as shown in **Fig. 1**. *SecurityAssets* contains entities that have a value, i.e. asset means an entity that has to be protected. *SecurityAttributes* contains concepts *Confidentiality*, *Integrity* and *Availability* – also called CIA triad. It must be noted that in telecommunications, this triad is often enhanced with *non-repudiation* and explicit reference to *authentication* and *authorization*. These security attributes compose service's security, and thus, security specification of a system should cover all of these aspects. After that, the concept *SecurityThreats* defines *Faults*, *Errors* and *Failures*. A failure is defined as

an event that occurs when the delivered service differ from the correct service. Instead, deviation in the external state of the system is called an error and the cause of an error is called fault. Internal faults are called vulnerabilities, whereas external faults are called attacks. The *SecuritySolutions* contains means for preventing unwanted behaviour and development of a system. The last concept is *SecurityMetrics* divided to *StrengthMetrics* and *WeaknessMetrics* – containing eight metrics to measure system's threats and efficiency of their countermeasures [7]. However, this categorization is at very high abstraction level.

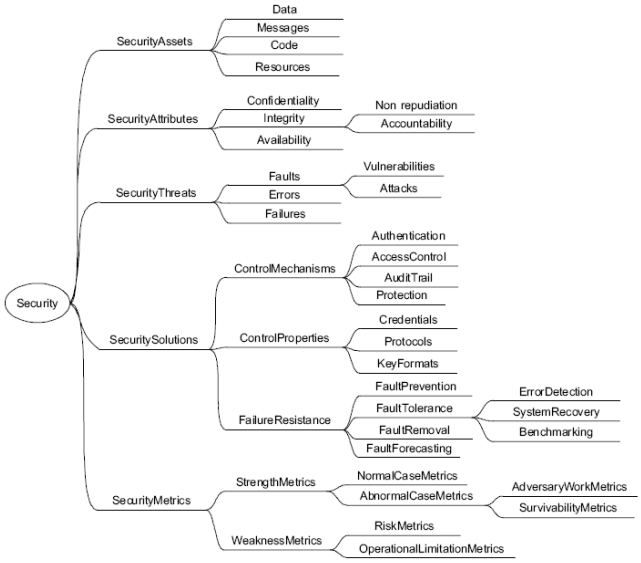


Fig. 1 Security taxonomy by Savolainen et al.

Denker et al. describe security annotations, represented by DAML-S, intended to be used by agents in [8]. Their ontology can be used to describe service requirements and capabilities – e.g. the requirement that a service requestor wants to use the Open-PGP encryption – and this information are used for service discovery and matchmaking. Matchmaking is performed based on requirements and capabilities with four matching level. [8]

Fig 2 presents the main parts of ontology from Denker et al. [8]. These ontologies contain *credentials information* and *security mechanisms*. Therefore, many important security aspects are missing. When compared to the work by Savolainen et al., for instance assets, threats and metrics are not defined. However, purpose of the Denker's ontology is to concentrate mostly to the service discovery and matchmaking, and thus

scope is different than ontology in [7]. Lately, Denker et al. updated their ontology to utilising OWL in [9].

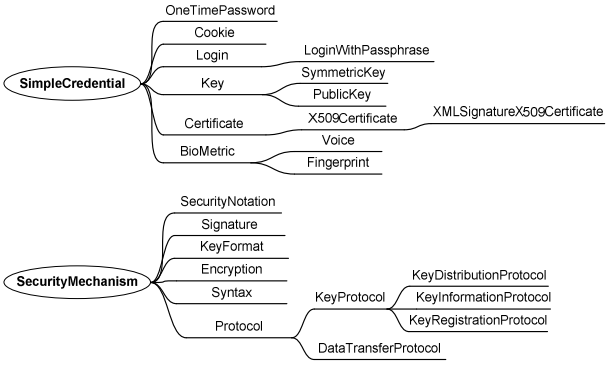


Fig 2 Security credentials and mechanisms by Denker et al.

Kim et al. presented their security ontology called NRL (Naval Research Laboratory) security ontology in [10]. NRL security ontology makes it possible to annotate resources with security related information that can be used during service discovery and matchmaking. NRL security ontology describes following security aspects: *mechanisms, protocols, objectives, algorithms* and *credentials* in various levels of details. Both service providers and requestors can use NRL ontology to describe their capabilities and requirements. Actually, NRL security ontology is a collection of ontologies. Main Security ontology imports the Credentials, Security Algorithms and Security Assurance ontologies as object properties. Thus, these ontologies make it possible to give more specific values for *SecurityConcepts*. In addition, user can define *SecurityObjectives* and connect them to the specific *SecurityConcept* by utilising *supportSecurityObjective* property. [10]

NRL security ontology is well organised concentrating mostly on security solutions area and providing a reasonable way to matchmaking between requirements and capabilities. When comparing NRL ontology to work by Savolainen et al. it can be noticed that the NRL security ontology lacks in security assets, threats and metric areas. However, NRL security ontology contains a substantially better description in security solutions area. In addition, connections between security objectives (requirements) and their solutions are defined more comprehensively.

Tsoumas et al. present in [11] and [12] a framework for information system security management, i.e. linking high-level policy statements to explicit low-level security controls adaptable and applicable in the information system environment. Authors' framework consists of four phases: 1) Building of Security Ontology, 2) Security Requirements Collection, 3) Security Actions Definition, and 4) Security Actions Deployment and Monitoring. Building of Security Ontology means ontology's instantiation based on a conceptual model. This conceptual model defines:

Asset, *Attack*, *Controls*, *Countermeasure*, *Impact*, *Security policy*, *Stakeholder*, *Risk*, *Threat*, *Threat agent*, *Unwanted incident*, *Vulnerability* and *relationships* among these concepts. When compared ontology from Tsoumas et al. to ontology from Savolainen et al. it can be noticed that both ontologies describe security in the same abstraction level. Thus, neither describes detailed protocols or mechanism for achieving security as opposed to NRL security ontology does. As well as NRL security ontology and ontology from Denker et al., ontology from Tsoumas et al. suffers from lack of security metrics.

Table 1 summarizes the most important aspects of above described security ontologies from run-time security management viewpoint. Ontologies from Denker et al. and Kim et al. are initially intended for run-time environment, i.e. service discovery and matchmaking. On the other hand, only ontology from Savolainen et al. contains security metrics. Thus, combining proposals from Savolainen et al. and Kim et al. should offer an appropriate approach. However, combining ontologies in an appropriate way contains many challenges. Firstly, suitable abstraction level has to be found. Secondly, relationships between metrics and security solutions are required, in order to measure solutions' efficiency during the run-time. In addition one of the major challenges is how to incorporate attacker behaviour to the ontology. Malicious activity is the major difference in security compared to other quality attribute descriptions.

Table 1 Summary of security ontologies

Ontology	Scope	Pros / cons
Savolainen et al.	Design phase of service centric systems.	+ Contains metrics – No connection between attributes and solutions – Only abstract level hierarchy – Intended for design-time use
Denker et al.	Service discovery and matchmaking	+ Run-time applicable – Only credentials and mechanisms.
Kim et al.	Service discovery and matchmaking	+ Run-time applicable + Detailed solutions + Connection between objectives and solutions – Metrics are missing
Tsoumas et al.	Linking policy statements to security controls	– Only abstract level hierarchy – Intended for design-time use

2.2 Security Measurement

It is a widely accepted management principle that an activity cannot be managed well if it cannot be measured. Overall, metrics provide four fundamental benefits – to characterize, to evaluate, to predict and to improve. Security metrics and measurements can be used for decision support, especially in assessment and prediction. Examples of using security metrics for assessment include [14]:

- Comparison of different security controls or solutions,
- Security assurance of a product, an organization, or a process,

- Security testing (functional, red team and penetration testing) of a system,
- Certification and evaluation (e.g. based on Common Criteria [15]) of a product or an organization,
- Intrusion detection in a system, and
- Other reactive security solutions such as antivirus software.

The field of defining security metrics systematically is young and the current practice of information security is still a highly diverse field; holistic and widely accepted approaches are still missing. A major challenge in developing appropriate and feasible security metrics is the immaturity of the state-of-the-art security requirements engineering. Security requirements have not been profoundly addressed within the software engineering community: they are still regarded as being in a side role in most of the software requirements engineering codes of practice [18].

In order to measure, the target of measurement needs to be identified. It is important to clearly know the entity that is the target of measurement because otherwise the actual metrics might not be meaningful. The target of security measurement can be, e.g., an organization, its processes and resources, or a product or its subsystem. The most widely known technical security certification standard is the Common Criteria (CC) ISO/IEC 15408 international standard [15]. During the CC evaluation process, a numerical rating, EAL (Evaluation Assurance Level), is assigned to the target product, with EAL1 being the most basic and EAL7 being the most stringent level. Each EAL corresponds to a collection of assurance requirements, which covers the complete development of a product with a given level of strictness.

ISO/IEC 9126 standard concentrates measuring software's quality, and it contains three metric reports: 1) External metrics [4] applicable during testing, 2) Internal metrics [5] applicable during development and 3) Quality in use metrics [6] applicable in run-time. Furthermore, ISO/IEC divides quality attributes into characteristics and subcharacteristics – security can be found under the functionality characteristic. Therefore, metrics from ISO/IEC are grouped to these characteristics and their subcharacteristics. However, standard does not contain any security related metric in Quality in use metric document [6].

Therefore, it is necessary to find a suitable way to measure security more extensively and holistically. Wang et al. presented a security measurement framework in [2] based on security requirements. The main idea in their work is to divide security requirements to smaller parts, i.e. decomposition, and finally these smallest parts can be measured. Garcia et al. propose two terms: a base measure and a derived measure in [19]. The base measure is a measure of quality attribute that does not depend on other measures, whereas the derived measure is a measure derived from base or derived measures. Therefore, base and derived measure terms can be combined to the approach of Wang et al. **Fig 3** shows the decomposition made for non-repudiation in [2] – combined with the base measure and derived measure terms from [19]. It can be noticed that the lowest level in **Fig 3** contains reliability – and metrics for it can be found, for example from ISO/IEC's standard. Thus, decomposition gives a possibility to find and develop security metrics required for the run-time security monitoring. When searching these metrics following sources can also give a valuable input: The U.S. NIST (National Institute of Information Standards and Technology) SAMATE (Software Assurance Metrics and Tool Evaluation) project [16] that seeks to help answer various questions on software

assurance, tools and metrics. OWASP [17] contains an active discussion and development forum on security metrics. More security metrics are listed in [14] and [18]. Representing these base and derived measures in the ontology also makes it possible to utilise measures at run-time, and in addition, generate new derived measures from the existing measures if needed.

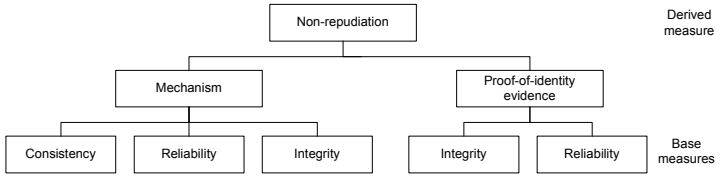


Fig 3 Decomposition for non-repudiation

3 Security Modelling

This section describes how to design security and how the process should take run-time security management issues into account in order to make it possible to achieve run-time security management.

3.1 Design-time Security Management

Our earlier work has concentrated to take quality issues into account at software's design and implementation phases by exploiting architectural models [20] and [21], i.e. conforming to model-driven development. We have used ontologies to describe quality attributes in a uniform way. Ontologies are utilized during the quality requirements modelling and during the architecture modelling to select the best solutions to achieve required qualities. Furthermore, we have evaluated a designed architecture in order to detect whether required qualities are met or not. Finally, quality of the implemented software is measured and compared to requirements. All of these phases belong to the QADA (Quality-driven Architecture Design and quality Analysis) methodology [23]. QADA contains two abstraction levels, i.e. conceptual and concrete levels, which can be mapped to the PIM (Platform-Independent Model) and PSM (Platform-Specific Model) from MDA.

Based on our earlier work, we are able to take step forward and start to concentrate to the run-time quality management – especially from security point of view. The means for defining quality attributes (especially reliability) at modelling-time is represented in [20] – and **Fig. 4** follows this approach from the security viewpoint. Thus at this point, the purpose is to model security requirements and the architecture in a way that makes it possible to achieve run-time security management.

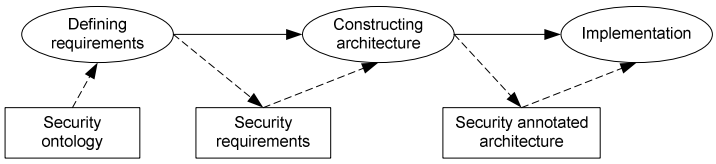


Fig. 4 Main phases of security modelling process.

Firstly, ontologies work as an input for the *Defining requirements* activity. Thus, ontologies described in the previous section have to be combined appropriately in order to achieve one adequate ontology – containing at least security objectives, countermeasures, base/derived measures and relationships between these concepts. After that, the security ontology can facilitate a requirements definition activity comprehensively, as a guideline. For instance, an architect can define following security requirements based on security objectives described in the security ontology and the objectives of the designed software: "user has to be authenticated" and "messages' integrity has to be ensured".

Constructing architecture activity starts after requirements definition. Support for run-time security adaptation requires that alternative security solutions are modelled and implemented into the system. The security ontology helps to find these alternative security solutions. Therefore, the architect selects from the security ontology a password and fingerprints as alternative solutions for the authentication requirement and AES (Advanced Encryption Standard) and DES (Data Encryption Standard) as alternative solutions for the integrity requirement. In addition, the architect selects appropriate measures from the security ontology to monitor achievement of the authentication and integrity requirements. Finally, we have security annotated architecture that works as an input for *Implementation* activity. Therefore, implementation can produce software containing alternative solutions for achieving security on different contexts, and measurements for ensuring that desired security level is kept.

In conclusion, the security ontology is used both defining security requirements and constructing architecture that contains alternative security solutions. Without these steps it is not possible to achieve run-time security management described in the next section.

3.2 Run-Time Security Management

Our current vision of the run-time security management is presented in **Fig. 5** – in other words, how the above designed and developed software monitors and adapts its security.

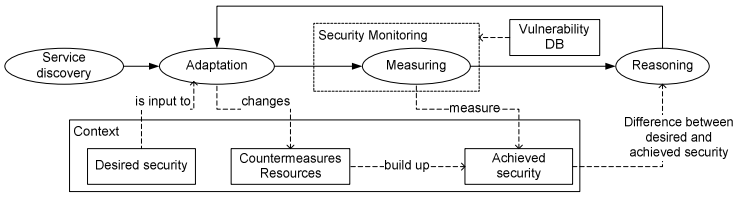


Fig. 5 Run-time security management

Firstly, available services are discovered. Thereafter, *Adaptation* is performed based on available services, their security properties, and the desired security level. In this phase, the security ontology helps to select the most suitable countermeasure (i.e. security solution as mentioned in the previous section) and resources in order to achieve the desired security level. Furthermore, this phase can utilize the existing service matchmaking solutions, but the adaptation also requires more sophisticated algorithms for the decision making. The purpose is to achieve an automatic adaptation that does not require user actions. However, user preferences can be used to direct a countermeasure selection or setting a divergent desired security level.

Next, the achieved security is *monitored* by means of measuring. A *Measuring* activity measures several properties of the system utilising base measures – introduced in **Fig 3**. From these values the *Monitoring* activity combines an overall security level by means of derived measures. Thus, monitoring activity also requires security ontology. Vulnerability databases can be utilised as an additional input source for monitoring activity. For instance, selected countermeasure might be cracked after ontology construction, and thus its security efficiency is lower than initially set. Hence, utilisation of vulnerability DBs enhances correctness of monitoring.

The achieved security value acts as an input for the *Reasoning* activity, which calls the *Adaptation* if the achieved security level is not satisfying the desired security and alternative countermeasure can be selected. Both the monitoring and the reasoning activities should be automatic – working without user actions. On the other hand, if the desired security level is not achieved and adaptation cannot resolve a situation then user actions will be needed to make a decision how to continue. In **Fig. 5** desired and achieved security levels are part of the context because the security management depends on the context where the adaptation occurs. How context is to be defined is out of the scope of this paper.

As a simple example of run-time security management, software contains AES / DES encryptions and fingerprint / username password pair authentication for achieving its security – as implemented in the previous section. We assume that a user prefers to use fingerprint authentication without encryption as default. In the case when content of the user's information exchange changes, for example, from news reading to more secure online buying, i.e. context changes, the desired security level also changes. The monitoring activity uses base and derived measures from the security ontology for monitoring the achieved security level. Based on the results of the monitoring, the reasoning activity remarks that the desired security level is not

reached anymore. Hence, the adaptation is called, and it decides that the security level for online buying cannot be satisfied without encryption and thus AES is selected – for instance.

4 Conclusions and Future Work

Managing security at the run-time requires that: 1) security is understood holistically yet offering enough details to be useful, 2) the achieved security level of the software can be monitored, and 3) software is able to adapt itself based on the context and monitoring results. In this paper, we concentrated to the first issue and also presented the overview vision how the run-time security management can be achieved.

Several security ontologies exist but mostly emphasize service discovery and matchmaking, or alternatively describe different security solutions. Although both are important aspects, these are not enough for covering the whole area of security from the run-time point of view. Thus, there is a need for more extensive security ontology. In addition, security ontology has to contain base and derived measures which make it possible to measure and monitor security levels.

Our next step is to produce the universal security ontology – a combination from the existing ones – containing security objectives and supporting solution mechanisms. Next, we will develop a set of base and derived measures for measuring efficiency of security solutions and include these metrics to the combined ontology. After that we are able to move forward to research monitoring mechanisms and adaptation algorithms required to support the whole process. When the first monitoring mechanisms are available we can build up an initial laboratory case to test validity of the approach and reveal it costs related the performance and other quality attributes.

Acknowledgments. This work is made under SOFIA (Smart Objects For Intelligent Applications) project – funded by Tekes (the Finnish Funding Agency for Technology and Innovation) and the European Commission.

References

1. Avižienis, A., Laprie, J-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing. Volume 1, Issue 1, pp. 11-33 (2004)
2. Wang, C., Wulf, W. A.: Towards a framework for security measurement. 20th National Information Systems Security Conference, Baltimore, pp. 522-533 (1997)
3. ISO/IEC: 9126-1 Software engineering – Product quality – Part 1: Quality model (2001)
4. ISO/IEC: 9126-2 Software engineering – Product quality – Part 2: External metrics (2003)
5. ISO/IEC: 9126-3 Software engineering – Product quality – Part 3: Internal metrics (2003)
6. ISO/IEC: 9126-4 Software engineering – Product quality – Part 4: Quality in use metrics (2004)

7. Savolainen, P., Niemelä, E., Savola, R.: A Taxonomy of Information Security for Service-Centric Systems. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 5--12 (2007)
8. Denker, G., Kagal, L., Finin, T., Paulucci, M., Sycara, K.: Security for DAML web services: Annotating and matchmaking. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, pp. 335-350 (2003)
9. Denker, G., Kagal, L., Finin, T.: Security in the Semantic Web Using OWL. Information Security Technical Report, Volume 10, Issue 1, pp. 51-58 (2005)
10. Kim, A., Luo, J., Kang, M.: Security Ontology for Annotating Resources. OTM Confederated International Conferences, CoopIS, DOA, and ODBASE. Springer, Heidelberg. pp. 1483--1499 (2005)
11. Tsoumas, B., Dritsas, S., Gritzalis, D.: An Ontology-Based Approach to Information Systems Security Management. In Computer Network Security, pp. 151-164 (2005)
12. Tsoumas, B., Gritzalis, D.: Towards an Ontology-Based Security Management. 20th International Conference on Advanced Information Networking and Applications AINA), pp. 985-992 (2006)
14. Savola, R.: Towards a Taxonomy for Information Security Metrics. In Proc. of ACM Workshop of Quality of Protection (QoP'07), Alexandria, Virginia, USA, pp. 28-30 (2007)
15. ISO/IEC: 15408-1 Common Criteria for Information Technology Security Evaluation – Part 1: Introduction and General Model. (2005)
16. Black, P. E.: SAMATE's Contribution to Information Assurance. IANewsletter, Volume 9, Issue 2 (2006)
17. OWASP: Open Web Application Security Project. http://www.owasp.org/
18. Savola, R., Abie, H.: Identification of Basic Measurable Security Components for a Distributed Messaging System. 3rd Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE), Jun 18-23, 2009, Athens, Greece (2009) in press
19. Garcia, F., Bertoa, M. F., Calero, C., Vallecillo, A., Ruíz, F., Genero M.: Towards a consistent terminology for software measurement. In Information and Software Technology, Volume 48, Issue 8, pp. 631-644 (2006)
20. Niemelä, E., Evesti, A., Savolainen, P.: Modeling Quality Attribute Variability: 3rd international conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 169-176 (2008)
21. Evesti, A., Niemelä, E., Henttonen, K., Palviainen, M.: A Tool Chain for Quality-driven Software Architecting. 12th International Software Product Line Conference (SPLC), p. 360 (2008)
22. Anderson, R.: Security Engineering – A Guide to Building Dependable Distributed Systems. John Wiley & Sons, New York (2001)
23. QADA (Quality-driven Architecture Design and quality Analysis). www.vtt.fi/qada/

PUBLICATION II

Towards micro architecture for security adaptation

In: Proceedings of the Fourth European Conference on Software Architecture (ECSA): Companion Volume, Copenhagen, Denmark, 23 August 2010. Pp. 181–188.

Towards Micro Architecture for Security Adaptation

Antti Evesti
VTT Technical Research Centre of Finland
Kaitoväylä 1, P.O. Box 1100
90571 Oulu Finland
+358 40 552 7542
Antti.Evesti@vtt.fi

Susanna Pantsar-Syväniemi
VTT Technical Research Centre of Finland
Kaitoväylä 1, P.O. Box 1100
90571 Oulu Finland
+358 40 505 6682
Susanna.Pantsar-Syvaniemi@vtt.fi

ABSTRACT
Normally, software development practices concentrate to take all security requirements into account at design-time. Nevertheless, today's software products are intended to be used in mobile, or alternatively, in embedded devices whose environment changes during the application's execution. These kinds of changes occur especially in applications used in smart spaces. This enforces to think security concerns more dynamically. Thus, software has to be aware of its 1) security level in each time, and 2) changes in its environment that can cause security threats. Based on this awareness, software has to change its security mechanisms to fulfil security requirements in the current context. A security measurement is a key factor of this awareness. This work presents a micro-architecture for security adaptation and taxonomy of context information affecting to information security in smart spaces. The security measurement is the essential part of the micro-architecture. In addition, taxonomy describes concepts that have to be monitored in the smart space environment.

General Terms
Management, Measurement, Design, Security.

Keywords
Quality, run-time, smart space.

1. INTRODUCTION
Our environment can contain a huge amount of sensors and devices. These can communicate and share information with each other, and thus, create different smart spaces to our surroundings. Software used in these kind of smart spaces encounter constantly changes in the environment and a way how the software is used. Thus, it is impossible to define beforehand all the changes in the smart space. For instance, an execution platform of an application may change from a laptop to a mobile phone, or alternatively usage of an application may change from entertainment to professional usage. Hence, it is infeasible to bind all security mechanisms at a design-time. Instead, an application, used in smart spaces, has to be able to monitor its environment and measure achieved security levels. Based on the environment monitoring and the security measurement the application has to make a selection of a security mechanism. In other words, the application has to have awareness of its situation.

ECSA 2010, August 23–26, 2010, Copenhagen, Denmark.

There are few approaches which concentrate to changing security requirements, or alternatively to adapt used security mechanisms to achieve the optimal security performance. Security Adaptation Management (SAM) [16] adapts security mechanisms of the system when a number of attacks increase. This makes it possible to reach more performance when attacks are not present. SAM addresses to buffer overflow attacks and countermeasures against them. The adaptation model presented in [2] concentrates on the tradeoffs between Quality of Service (QoS) and security items. The purpose of the presented model is to select the most suitable application variant for each situation based on context information and user preferences. However, these approaches cannot offer enough situation awareness and autonomy needed for the adaptations of smart spaces.

In this work we present an adaptation approach that offers required flexibility and autonomy especially for smart spaces – by utilising context monitoring and security measuring. A context change is a trigger for the adaptation. The context change means either change in the physical or digital environment of an application or a change in the usage of the application. Based on these changes the application makes a decision on its required security level. After that, the application (or supporting services) measures if the requirement is met or not. Finally, the application adapts the used security mechanisms and parameters if the desired security level cannot be met with the used mechanisms. In this approach, ontologies are in the key role – exploited two different purposes. Firstly, ontology is used to describe a security in general. The security ontology describes how the fulfilment of a security requirement can be measured. Furthermore, the security ontology contains security goals and supporting mechanisms. Secondly, ontology is used to describe context information affecting to security of the smart space application. Novelty of this work comes from the utilisation of ontologies, which enables an application to adapt to several situations. New knowledge can be brought to the application afterwards with the ontologies. This new knowledge can be, for example, information of arisen security vulnerabilities and new security measurement technique.

The paper is organised as follows: Section 2 gives background information. Section 3 describes the taxonomy of security context information and section 4 describes the micro architecture for

achieving security adaptation in the smart spaces. Finally, discussion and conclusions close the paper.

2. BACKGROUND

In this section, the used smart space platform is presented. Secondly, security ontologies are briefly presented. After that, context modelling, situations and related ontologies are described. Finally, section 2.5 introduces the context monitoring approach and security monitoring that will be combined in this work.

2.1 Smart-M3

In this work, the smart spaces are built up by using a Smart-M3 platform [1], and thus, related terminology will be used. The purpose of the Smart-M3 is to make the information available between heterogeneous devices and add semantics for this information. In the Smart-M3, a SIB (Semantic Information Broker) creates a backbone to the smart space. A KP (Knowledge Processor) works as an independent agent producing and/or consuming information from the SIB. Thus, all the information is transmitted via the SIB, i.e., KPs do not communicate directly with each other. The information transmission is performed by a SSAP (Smart Space Application Protocol). The SSAP can be utilised with different communication protocols, like TCP, Bluetooth.

2.2 Security Ontologies

Zhou defines ontology as a shared knowledge standard or a knowledge model defining primitive concepts, relations, rules and their instances, which comprise topic knowledge. The ontology can be used for capturing, structuring and enlarging explicit and tacit topic knowledge across people, organizations, and computer and software systems [25].

There are several security ontologies available for different use. Blanco et al. offer a list of security ontologies in [5]. In addition, [12] compares few security ontologies from a run-time applicability viewpoint. Security taxonomy from Savolainen et al. presents a hierarchy of security concepts – intended especially for a design time usage [21]. That taxonomy is the only one containing at least some kind of presentation of security measurements. The NRL (Naval Research Laboratory) security ontology presented by Kim et al. [17] gives an extensive set of security concepts and base relationships between concepts. The NRL security ontology is intended for service discovery and matchmaking purposes – referring to run-time applicability. Herzog et al. present in [15] an ontology of information security called OIS in this work. The OIS contains a comprehensive set of security concepts divided to countermeasures (133), assets (79), vulnerabilities, and threats (88), and in addition relationships between those concepts (34) – numbers in parenthesis mean a number of pieces. The OIS can work as a general purpose vocabulary, an extensible dictionary, or a manner of reasoning relationships between concepts, like threats and countermeasures as authors mention. Therefore, the OIS offers a good starting point for ontology required in the run-time security management. However, it does not contain security measurements which are essential to measure the achieved security level. Hence, security measurements part has to be added to the OIS. However, this ontology development is out of scope of this paper.

2.3 Context Modelling and Situations

Chen and Kotz give the following definition for context: Context is a set of environmental states and settings that either determines an application's behaviour or in which application event occurs and is interesting to the user [6]. Bettini et al. [4] describe that ontological models have clear advantages regarding support for interoperability and heterogeneity. Besides, the ontological models support the representation of complex relationships and dependencies between context data. Thus, the ontological models are well suited to the recognition of high-level context abstractions. A situation is the mostly used term for referring high-level context abstractions.

Situations as a concept are introduced in [8, 9]. Dey et al. introduce in [10] the usage of situations to support intelligibility and control in context-aware applications. In [10], the situations are components that expose application logic. Dobson et al. in [11] describe situations as external semantic interpretations of low-level context. Adaptations in context-aware applications are caused by the change of situations, i.e., a change of a context value triggers adaptation if the context update changes the situation [4]. Bettini et al. present in [4] an overview of the different levels of semantic context interpretation and abstraction. In that overview sensor-based low-level context information (1st level) is semantically interpreted by the high-level context level (2nd level). The situations (3rd level) are formed based on the information that is reusable and available from the 2nd level. In [4] it is defined that the situations can either be defined by a human based on his knowledge or recognized and learned automatically by using machine learning techniques.

2.4 Context Ontologies

For the run-time security adaptation in the smart spaces we need to have context ontology beside the security ontology. CoDAMoS (Context-Driven Adaptation of Mobile Services) context ontology [20] is user-centric since users play an important role in ambient intelligence and covers context-awareness, situation awareness, and domain independency. The CAMPO [24] ontology is to be used with the CAMPUS (Context-Aware Middleware for Pervasive and Ubiquitous Service) middleware that utilizes the CAMPO to dynamically derive decision rules to enable optimized computation over varying contextual environments and covers context-awareness, and domain independency. The CAMPO is a component-centric ontology since it is used in CAMPUS to decide whether the component can be used in the target environment. The SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) ontology [7] is intended for pervasive computing and used for building the CoBrA (Context Broker Architecture) middleware. It covers context-awareness, situation awareness, domain independency, and adaptability rules. The SOUPA defines intelligent agents with associated beliefs, desires, and intentions, time, space, events, user profiles, actions, and policies for security and privacy. The SOUPA ontology is attractive for our purposes since it is agent-centric and aimed for the smart spaces.

2.5 Related Work

Toninelli et al. [23] present the context-aware middleware architecture for smart spaces. It is an extension of Smart-M3 introducing two alternative implementation options for building up context-aware support for the smart spaces. This context-aware

middleware architecture is used as basis for the context ontology and a concept of context-awareness that takes care for instance of context monitoring [19]. This context ontology is created by OWL (Web Ontology Language) by importing some parts from the SOUPA [7] and by adding own domain related classes. The main classes of the context ontology (user, digital environment, and physical environment) follow the definition given in [22]. This context ontology takes into account the different levels of semantic context interpretation and abstraction presented in [4]. The first level, the physical context of environment, detects individual actions. Thereafter, the second level – the digital context of environment – fuses the individual output of the first level. Finally, the third level fuses the output of the second level for a group of situations. These terms are utilised also in this work when context information for security is defined.

An initial vision of run-time security monitoring is presented in [12]. The presented model contains service discovery, adaptation, measuring, and reasoning activities as a sequence with a feedback loop. In the model, adaptation is performed immediately after service discovery. With the term adaptation we refer to the ability of software to adapt its functionality, especially its security mechanisms, according to the current context, refined from adaptation definition in [18]. Thus, the purpose of the adaptation action is to select the most optimal security mechanisms based on the current context information. After that, achieved security is monitored by means of measuring. An achieved security value is an input for the reasoning activity, which will call the adaptation again if the desired security level is not reached.

In this work, we will draw a connection between approaches described above. Thus, the initial vision of run-time security monitoring is enhanced. In addition, the used context information for security is bound to the context levels presented in [19].

3. CONTEXT INFORMATION FOR SECURITY

Several context ontologies exist as mentioned in the previous section. However, the context information is thought from a security viewpoint in this work. Consequently, we have to define the context concepts that affect to the security of application in the smart space. In other words, these concepts affect to the required security level and also to the achieved security level. Thereafter, we map these concepts to three context levels, i.e., situation, digital, and physical context, as depicted in Figure 1.

In the highest abstraction level, in the situation context, are the concepts that describe an application usage: 1 *Role of exchanged / stored data* and 2 *User's role in a smart space*. The role of data, either exchanged or stored, contains a content type property and content usage property. The content type property has values like a picture, a video, an e-mail, and a text message. The content usage property has the value of entertainment, professional, payment, or emergency. In addition, data can have a level of importance, privacy, etc. The concept *user's role in a smart space* gets values like a visitor, a worker, a customer, an owner, etc. In addition, each of these user roles can have own objectives in the smart space. As a conclusion, above mentioned context concepts are intended to describe the usage of smart space application and its user. Hence, this information is utilized to decide security requirements and levels for the application in the smart space.

On the contrary, context information of the environment of an application affects to the ability to reach the security requirements and levels. The context information of environment is divided into two levels – digital and physical context – as mentioned in the previous section. From the security viewpoint, the digital context means a *role of smart space* and the physical context means an *execution platform* of smart space application. The *execution platform* concept describes a platform where the smart space application is executed – containing a connection type, an operating system, supported security mechanisms, etc.

The *role of smart space* concept is intended to describe the smart space where the application is currently used, i.e., private, office, or public. The SIB and the KPs build up the smart space as explained in the section 2.1. Thus, the reputation of SIB and KPs is important context information from the security perspective. The reputation consists of information of KPs and SIB producers, utilization of an antivirus software, and used authentication, etc. The producer of KPs and SIB tells to the application that can it utilize the information from the particular agent or not and is it allowed to join the particular SIB. For example, a user is able to set a preference that applications in her device utilize only the KPs and the SIBs produced by big companies. Similarly, joining and using information from smart spaces which doesn't have appropriate antivirus software or an authentication mechanism can be restricted.

From these context concepts – on three different levels – it is possible to make connections to the security ontology. From the platform concept connection to vulnerabilities and security mechanisms, i.e., countermeasures, can be performed. For instance, the operating system may contain vulnerabilities and it can also offer security mechanisms. Secondly, used authentication mechanism can be connected to the security ontology. Finally, a connection to assets in the security ontology can be made from the *role of exchanged / stored data* concept.

To collect all this information by a means of context monitoring is unfeasible. Thus, the information that cannot be collected automatically has to be collected from the user – for instance by a means of user preferences. The above described context information is utilised during the security adaptation that will be described next.

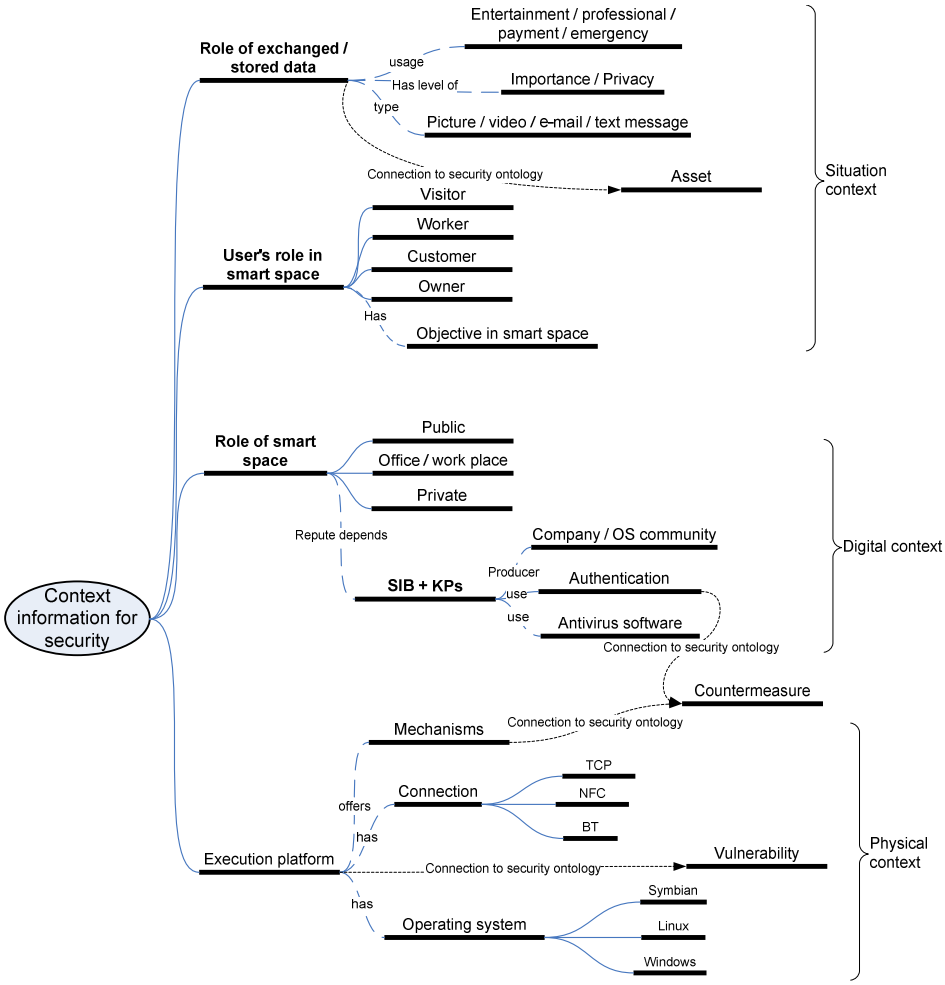


Figure 1 Context information for security

4. MICRO ARCHITECTURE FOR SECURITY ADAPTATION

In this section we describe micro architecture for adaptable security, based on [12]. Figure 2 presents designed micro architecture, containing six execution phases (rectangles) and the most important information (three-dimensional rectangles) flowing between execution phases. The execution phases are:

1) Decision of required securities and levels – decides the required securities and levels in the current context situation.

2) Retrieve supporting mechanisms from ontology – searches the security mechanisms that can support requirements.

3) Control analysis – analyses which security mechanisms can be used in the current smart space.

4) Measuring and Reasoning – measures the achieved security level in the current environment and reasons the affects of security threats. Applicable measures are searched from the ontology.

5) Execution of Adaptation – selects the security mechanism and parameters that offer the optimal security level in the current situation. After the execution of adaptation, execution phase 1 or 4 will be executed depending on the input from phase 6.

6) Context monitoring – produce the information defined in the context information for security taxonomy for the execution phases 1 and 4.

Briefly, there are two factors causing a need for security adaptation. Firstly, the level of security requirement may change during the execution, recognised by a means of context monitoring. Secondly, the application's capability to achieve the required security level may change due to the change in the environment, recognised by a means of measuring. Therefore, this model deals both of these cases.

4.1 Decision of Required Securities and Levels

The first phase recognises changes in the security requirements and levels. Naturally, the application is unable to create the security requirements from scratch but a parameter selection can be made automatically without major contribution from the user. Firesmith notes that there is not much variation in the security requirements between the different applications [13]. Based on this, Firesmith proposes a security requirements template to define reusable security requirements. The requirements decision phase works in a similar way for deciding the security requirements and levels of each situation.

The first step in the requirements decision phase is asset recognition – in this work an asset is data stored in a device or in a communication channel. However, content, usage, and user of this data vary, as depicted in Figure 1. Consequently, this variation affects importance of data and criticality of threats. Therefore, the context monitoring is required to recognise the changes in *Role of exchanged / stored data* and *User's role in smart space* context concepts. The requirements decision phase subscribes to the results of context monitoring in the SIB, and thus, gets a notification when the situation context changes.

The output of the requirements decision phase is called a protection profile for the application – which bases on a current situation context. The protection profile is a platform independent way to present the security requirements of applications [3]. The protection profile describes which information has to be protected and how much. In this work we utilise five quantitative levels for describing the different amount of security, i.e., levels 1-5. The level 1 means that achieving the security requirement is not critical, and on a contrary the level 5 means that achieving the security requirement is extremely critical.

4.2 Retrieve Supporting Mechanisms from Ontology

The supporting security mechanisms are retrieved from the security ontology based on the content of the protection profile. In this phase, the mechanisms are searched only by the security requirement, not with a level value. The level depends on the current environment and it will be estimated for the mechanisms later on. The OIS described by Herzog et al [15] contains the connection from the requirements to the supporting countermeasures. However, retrieving the ontology returns a set of security mechanisms supporting desired security requirements. In addition, the ontology offers information related to which kind of asset a particular mechanism is intended to protect. Thus, for instance, different mechanisms may be offered for achieving confidentiality of data stored in the device or confidentiality of data in the communication channel.

4.3 Control Analysis

The control analysis phase compares and evaluates the feasibility of security mechanisms retrieved from the security ontology. This phase is required because the security ontology is generic and contains the mechanisms not supported by the application or not applicable at the current situation. The control analysis checks currently used security mechanisms – this information is required in the adaptation phase.

Secondly, the security mechanisms supported in the user's own device and application in it are found out. Finally, the mechanisms supported in the current smart space, i.e., the mechanisms that the currently used SIB and KPs can use are found out. In other words, it is not enough to know the security mechanisms supported in own device but also the mechanisms in other devices which the user's device is communicating with. Hence, the result of the control analysis is a reduced version of the security mechanisms list returned from the security ontology in the previous phase. This list is called to applicable mechanisms.

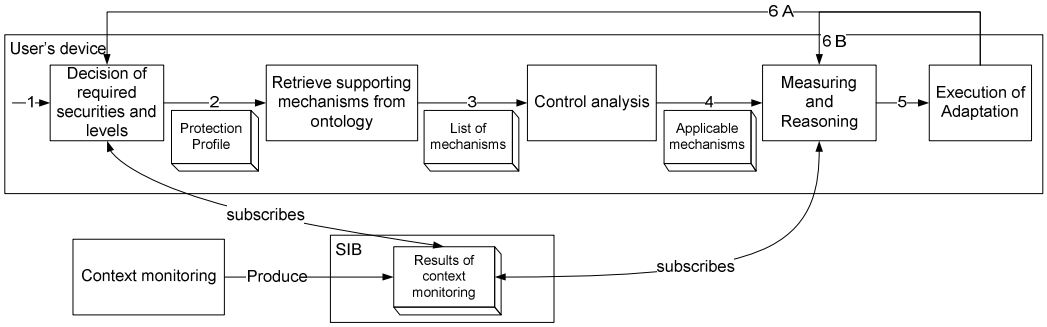


Figure 2 Micro Architecture for security adaptation

4.4 Measuring and Reasoning

The measuring phase is twofold. Firstly, the security level that different mechanisms can offer in the current environment is estimated. This means an estimation before the security mechanism is started to use. On the other hand, requirements fulfilment is measured during the application usage. This means the measuring which reveals if the used security mechanism is not

able to fulfil the required security in the changing environment. In order to obtain the required input information the measurement phase also subscribes to the result of context monitoring.

In this work, we utilise the security measurements from the risk management area. However, other security measurements can be also utilised, e.g. measures related to the mechanism correctness. The risk level is calculated by multiplying threat likelihood with severity of consequences [14].

*Risk Level = Threat Likelihood * Severity of Consequences*.

This calculation produces values in interval [0, 100] as shown in [14]. These values can be connected to the quantitative security levels, for instance as follows:

- *Risk Level* value 0-9 connects to the security level 5 (highest security)
- *Risk Level* value 10-19 connects to the security level 4
- *Risk Level* value 20-39 connects to the security level 3
- *Risk Level* value 40-69 connects to the security level 2
- *Risk Level* value 70-100 connects to the security level 1

Usage of exchanged / stored data (see Figure 1) affects to the *Severity of Consequences* variable. In this variable bigger values mean the greater damage. Hence, following *Severity of Consequences* values are initially suggested for different usage:

- For entertainment usage *Severity of Consequences* 5-8,
- For professional usage *Severity of Consequences* 25-50,
- For payment usage *Severity of Consequences* 40-65, and
- For emergency usage *Severity of Consequences* 80-100.

The variation between these values comes from the different usage inside the usage group. For instance, *Severity of Consequences* in the professional usage depends on type of business and in the payment usage money amount affects.

The second factor in the *Risk Level* calculation is *Threat Likelihood*. We have not developed the measurement techniques applicable for *Threat Likelihood* measuring at run-time so far. Therefore, reasoning, based on the environment information, is utilised to decide *Threat Likelihood* for the security risk level calculation. In this time following aspects are taken into account when reasoning *Threat Likelihood*: unknown devices in the smart space, founded vulnerabilities in the used security mechanism, and reliability or unreliability of the used security mechanism. Following table shows how these aspects affect to *Threat Likelihood* variable in the different smart spaces. However, these values are intended to be as initial values, which can be adjusted during the application execution.

Table 1 Affects to threat likelihoods in different smart spaces

	Home smart space	Work smart space	Public smart space
Unknown device in smart space	+ 50 percentage units	+ 20 percentage units	NULL
Founded vulnerability in the used security mechanisms	+ 10 percentage units	+ 25 percentage units	+ 50 percentage units
Reliability / unreliability of the used security mechanism	+ / - 25 percentage units	+ / - 25 percentage units	+ / - 25 percentage units

Now, these Risk Levels are described in a generic way. However, from the security adaptation point of view it is necessity to calculate levels separately for each security goals. Meaning the own risk level for authentication, integrity, etc.

4.5 Execution of Adaptation

The execution of adaptation phase performs the security mechanism selection and sets parameters for the selected mechanism. The execution of adaptation utilises the information collected from the previous phases:

1) the protection profile, describing the required securities and levels,
2) a list of applicable and currently used security mechanisms, and
3) the measurement and reasoning results.

Based on this information set, the execution of adaptation tries to select an optimal set of security mechanisms. For instance, the protection profile defines that an integrity level 2 is required, and the already used security mechanism is able to ensure the level 1 only. Thus, the execution of adaptation selects the new mechanism or changes parameters for the currently used mechanism. There is a possibility that adaptation cannot ensure the required security level, and then a warning message will appear to the user. Another alternative is that the security mechanism violates other qualities, like a performance, and then the execution of adaptation will require trade-off analysis, which is out of scope of this paper.

4.6 Context Monitoring

The context monitoring phase produces the context information to the SIB. Thus, it offers important input information for the requirements decision and for the measuring and reasoning phases. These two phases subscribe to the information produced by the context monitoring. Thus, the context change triggers the requirements decision and the measuring and reasoning phases. The context monitoring is the KP used by the smart space application or another KP in the smart space application to follow up the context information relevant for it. The scope is defined by another agent in the smart space or by the smart space application.

The context monitoring agent does not need to be located in the same computing environment than the other agent or smart space application who consumes the information provided by the context monitoring agent. The context information is mediated via the SIB by using a suitable communication channel between the SIB and the context monitoring KP and between the SIB and the

context information consumer. In this work the context information consumer refers to the requirements decision and the measuring and reasoning phases.

5. DISCUSSION

The presented micro architecture and the context taxonomy for security are in the research phase. We are currently implementing a demonstrator containing home and public smart spaces in order to validate how the presented adaptation approach and the related context monitoring and security measurement work in practice. The demonstration will also contain measurements related to security correctness, described by a means of ontologies. The security measurement is the most essential part in order to achieve the applications which are really able to autonomously adapt themselves in the smart spaces. Secondly, we will continue research related to the context monitoring. After these, we are able to take into account other quality attributes also, like performance, and related trade-offs.

In this time, we have three main issues in our minds. Firstly, measuring the achieved security level means the simplification of application and its environment. Hence, measuring security of the application does not give an exact truth – for example if comparing to the performance measurements like memory consumption. We utilise the security levels from 1 to 5 to describe the extent of security. These levels are derived from the risk management approaches, which utilise the threat likelihood and the severity of consequences to calculate the risk values. Developing measurement techniques which are able to measure these variables from the application and its environment at run-time is required. In this work, the threat likelihood is reasoned from the context information, i.e., unknown device in the smart space, founded vulnerability in the used security mechanism, and reliability / unreliability of the used security mechanism. The type of smart space affects how much these affect to the threat likelihood. Hence, we estimated effects in the home, work, and public smart spaces. Nevertheless, several other factors also affect to the threat likelihoods, for instance an attacker's capabilities and resources, and the severity of consequences because it may be a motive for the attacker. Furthermore, it is difficult to give precise values for the risk value calculation, and thus, using the security levels instead of exact values is reasonable. However, a calibration of these levels will be important.

Second issue relates to the context monitoring. The purpose of context monitoring from the security adaptation and measurement viewpoint is to offer the information needed to find out the required and achieved security levels. The defined taxonomy of security related context information specifies this context information. For a certainty, there exist other concepts that also affect the security levels of the smart space application. However, the most extensive set of security related context concepts will be found by testing the presented approach in the real smart spaces.

Thirdly, deciding the security requirements and levels for application based on the context information is not a trivial task. Deciding required security goals and levels needs a lot of reasoning. The smart space application has to have detailed awareness of its usage when selecting the security requirements and levels. Thus, currently used usage classifications (entertainment, professional. paying, and emergency) has to be defined further. For instance, the professional usage may contain several business domains. These business domains have different requirements for confidentiality, authentication, etc. However, we suppose that this phase will get benefit from the measuring techniques – because measuring is intended to measure the fulfilment of requirements.

There exist few approaches for security adaptation, i.e., SAM (Security Adaptation Management) [16] and the adaptation model presented in [2]. SAM adapts used security mechanisms when a number of attacks increase. In other words, the adaptation is performed reactively. In our approach, the adaptation occurs proactively. Proactiveness can be achieved by utilising the reasoned threat likelihoods in the security level measurements, i.e., threats appear before attacks. Using the attacks as a trigger for the adaptation might cause that the adaptation is performed too late. The adaptation model presented by Alia et al. collects the security requirements from user preferences. This is not practical in the smart spaces where the situations change every now and then. Enforcing the user to give new preferences all the time, or alternatively, using the same preferences in the different situations causing a need to overestimate the required securities. Therefore, our approach contains the requirements decision phase to decide the security requirements and levels for each situation.

In the future, autonomous behaviour of the smart spaces also affects to the achieved security. When devices and applications start to communicate with each other in an unexpected way – not defined earlier – new security problems might occur. In many cases when two secure products are combined the result has contained new security vulnerabilities. These problems have happened even though combination is well designed. Thus, in cases when the smart space applications perform these combinations autonomously it can be assumed that unexpected security vulnerabilities will occur.

6. CONCLUSIONS

In this paper we presented the taxonomy of context information for security. The taxonomy contains concepts describing: the usage of smart space application, the smart space itself, and the physical features of the environment. In addition, the micro architecture for adaptable security is modelled. The micro architecture contains six execution phases: requirements decision, retrieving security mechanisms from the security ontology, control analysis, measuring and reasoning, execution of adaptation, and context monitoring. The context monitoring phase produces input information for the requirements decision and the measuring and reasoning phases.

The ontologies play the key role in our approach. The security ontology is used to search the supporting security mechanisms for different situations. Moreover, the concepts affecting these situations are defined by a means of context taxonomy. The ontologies ensure that it is possible to bring new information related to new vulnerabilities in some security mechanism, a new measuring technique or alternatively, new context information affecting to the security of smart space application.

7. ACKNOWLEDGMENTS

This work has been carried out in the Sofia ARTEMIS JU project funded by Tekes, VTT and the European Commission.

8. REFERENCES

[1] Smart-M3 URL: http://sourceforge.net/projects/smart-m3/ Accessed: 06/30, 2010.

[2] Alia, M. and Lacoste, M. 2008. A QoS and security adaptation model for autonomic pervasive systems. In *32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008.* (Turku, 28 Jul. - 1 Aug. 2008), 943-948.

[3] Anderson, R. J. 2008. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, Indianapolis.

[4] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. 2010. A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing, 6, 2, 2010, 161-180.

[5] Blanco, C., Lasheras, J., Valencia-García, R., Fernández-Medina, E., Toval, A., and Piattini, M. 2008. A systematic review and comparison of security ontologies. In *3rd International Conference on Availability, Security, and Reliability, ARES 2008.* (Barcelona, 4 - 7 Mar. 2008), 813-820.

[6] Chen, G. and Kotz, D. 2000. *A Survey of Context-Aware Mobile Computing Research.* Technical Report TR2000-381. Dartmouth College.

[7] Chen, H., Finin, T., and Joshi, A. 2005. The SOUPA Ontology for Pervasive Computing. Ontologies for Agents: Theory and Experiences, 233-258.

[8] Dey, A. K. and Abowd, G. D. 2000. CybreMinder: A Context-Aware System for Supporting Reminders. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing.* Bristol, UK. Springer-Verlag, 172-186.

[9] Dey, A. K., Abowd, G. D., and Salber, D. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 16, 2, 2001, 97-166.

[10] Dey, A. K. and Newberger, A. 2009. Support for context-aware intelligibility and control. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems.* (Boston, 4-9 Apr. 2009). ACM, New York, 859-868.

[11] Dobson, S. and Ye, J. 2006. Using fibrations for situation identification. In *Pervasive 2006 workshop proceedings.* 645-651.

[12] Evesti, A., Ovaska, E., and Savola, R. 2009. From security modelling to run-time security monitoring. In *European Workshop on Security in Model Driven Architecture (SECMDA).* (Enchede, 23 - 26 Jun. 2009). CTIT Centre for Telematics and Information Technology, 33-41.

[13] Firesmith, D. 2004. Specifying reusable security requirements. Journal of Object Technology, 3, 1, 2004, 61-75.

[14] Herrmann, D. S. 2007. *Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI.* Auerbach Publications, Boca Raton.

[15] Herzog, A., Shahmehri, N., and Duma, C. 2009. An ontology of information security. Techniques and Applications for Advanced Information Privacy and Security: Emerging Organizational, Ethical, and Human Issues, 278-301.

[16] Hinton, H., Cowan, C., Delcambre, L., and Bowers, S. 1999. SAM: Security Adaptation Manager. Proceedings of 15th Annual Computer Security Applications Conference 1999 (ACSAC '99), (Phoenix, 6-10 Dec. 1999), 361-370.

[17] Kim, A., Luo, J., and Kang, M. 2005. Security Ontology for annotating resources. In *OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005 - On the Move to Meaningful Internet Systems 2005.* (Agia Napa, 31 Oct. - 4 Nov. 2005), 1483-1499.

[18] Matinlassi, M. and Niemelä, E. 2003. The impact of maintainability on component-based software systems. In *29th Euromicro Conference.* (Belek-Antalya, Turkey, 3 - 5 Sep. 2003), 25-32.

[19] Pantsar-Syväniemi, S., Simula, K., and Ovaska, E. 2010. Context-awareness in smart spaces. First International Workshop on Semantic Interoperability for Smart Spaces, (Riccione, Italy, 22 Jun. 2010), 1010-1015.

[20] Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., and De Bosschere, K. 2004. Towards an extensible context ontology for ambient intelligence. Lecture Notes in Computer Science, 2004, 148-159.

[21] Savolainen, P., Niemelä, E., and Savola, R. 2007. A taxonomy of information security for service centric systems. In *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2007.* (Lubeck, 27 - 31 Aug. 2007), 5-12.

[22] Soylu, A., Causmaecker, P. D., and Desmet, P. 2009. Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. Journal of Software, 4, 9, 2009, 992-1013.

[23] Toninelli, A., Pantsar-Syväniemi, S., Bellavista, P., and Ovaska, E. 2009. Supporting context awareness in smart environments: a scalable approach to information interoperability. In *M-PAC '09: Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing.* (Urbana Champaign, Illinois, 30 Nov. 2009). ACM, 1-4.

[24] Wei, E. J. Y. and Chan, A. T. S. 2008. Semantic Approach to Middleware-Driven Run-Time Context-Aware Adaptation Decision. In *ICSC '08: Proceedings of the 2008 IEEE International Conference on Semantic Computing.* (Santa Clara, CA, USA, 4-7 Aug. 2008). IEEE Computer Society, 440-447.

[25] Zhou, J. 2005. Knowledge Dichotomy and Semantic Knowledge Management. Industrial Applications of Semantic Web, (Jyväskylä, 25 - 27 Aug. 2005), 305-316.

PUBLICATION III

Ontology-based security adaptation at run-time

In: Proceedings of the Fourth IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO), Budapest, Hungary, 27 September – 1 October 2010. Pp. 204–212.

Ontology-based Security Adaptation at Run-time

Antti Evesti and Eila Ovaska
VTT Technical Research Centre of Finland
Finland
e-mail: antti.evesti@vtt.fi, eila.ovaska@vtt.fi

***Abstract*—This paper describes how software is able to autonomously adapt its security mechanisms based on knowledge from security ontology. Security adaptation is required because a software's environment changes during run-time. Thus, all security requirements cannot be defined beforehand. To achieve security adaptation, we have combined a security ontology that defines security mechanisms, security objectives, and high level security measurements. The run-time security adaptation utilises this security ontology to adapt security mechanisms or their parameters to fulfil security requirements for each environment and usage situation. The novelty of this approach comes from the utilisation of ontologies and security measurements, which makes adaptation flexible. We validate our security adaptation with a case study in a smart space environment. The case study proves that security adaptation is able to work autonomously without other user actions.**

Keywords-component; Security ontology, dynamic adaptation, quality management

I. INTRODUCTION

Nowadays, it is not enough to decide the security mechanisms used only at the time of a software's design. Present requirements demand that the software product is able to select the most suitable security mechanisms at run-time. This is so because many applications are executed in a constantly changing environment and it is therefore not possible to define all of these changes beforehand. In addition, all security mechanisms are not applicable in all possible environments, i.e. security mechanism is not supported or mechanism requires too much calculation capacity – causing a need to adapt application at run-time. Mobile devices used in smart environments face this kind of situations in particular. Although, the environment changes a user wants to preserve the particular security level. Therefore, run-time security adaptation ensures that the user is able to concentrate to application usage – instead of configuring its security features constantly. Moreover, an application with adaptation capabilities is able to offer increased performance because security mechanisms are used only when needed – not all the time.

Few approaches exist where changing security requirements and attack situations are the focal point. In [1] Security Adaptation Management (SAM) is presented. The purpose of SAM is to allow more performance and a less secure state to the system when it is not under an attack. When the number of attacks increases the SAM adapts the system to use different implementations to achieve a more secure state. This work utilises the term adaptation spaces for defining possible security breaches and related countermeasures. SAM concentrates on buffer overflow cases and defends against them. Alia et al. present an adaptation model for handling trade-offs between QoS (Quality of Service) and security concerns in [2]. The adaptation model consists of a component framework model, the context in a general sense, adaptability dimensions, user's preferences, and a utility function for describing an adaptation strategy. The utility function selects which application variant will be used in each situation – the selection is based on the user's preferences and context information. Lamprecht et al. presents a Secure Socket Layer (SSL) based run-time security adaptation in [3, 4] – called an adaptive SSL. The adaptive SSL selects an appropriate security mechanism for SSL session, i.e., performs a renegotiation, when the environment changes. The main interest of the authors is security adaptation from the resource consumption and performance viewpoint. However, a threat level is also mentioned as a possible trigger for the adaptation. Myllärniemi et al. present in [5] a security variability approach called KumbangSec – which is intended to work at an architectural level. With this kind of approach the purpose is to derive a different product for different security requirements. Therefore, the security mechanism is bound at design-time and changes are not possible afterwards. Hence, this kind of approach is insufficient in those cases when the security requirements of an application or security threats change during run-time.

In this paper we present our work towards applications which are able to adapt their security mechanisms at run-time. Security is a complex area, containing several bindings to other quality attributes, like reliability and performance. In addition, some security objectives are contradicting, like non-repudiation and privacy. These complexities are reason to utilise ontologies to describe security. Our adaptation approach is based on a security ontology with measurements, and context information representing changes in the environment and usage of application. The novelty of our approach comes from the utilisation of ontologies and security measurements, which ensures flexible and autonomous adaptation also in resource-restricted systems – like mobile devices used in smart spaces. In addition, the approach takes an asset's value into account that reflects directly to the security requirements of an application. The ontology is used to describe security mechanisms, their properties and supported security objectives, whereas security measurements are triggers for the security adaptation. In this work, we adopt security measurements

978-0-7695-4232-4/10 $26.00 © 2010 IEEE
DOI 10.1109/SASO.2010.11

from risk management approaches. Furthermore, utilisation of context information makes it possible to recognise different security requirements for the same application depending on its usage. This is essential when the same application is used for different purposes. Moreover, the approach is applicable for different threats.

After the introduction, background information is given. Thereafter in section 3 parts of the used security ontology and security measurements are presented. Section 4 presents our adaptation approach. Section 5 gives a case example and section 6 contains discussion and future work ideas. Finally, conclusions close the paper.

II. BACKGROUND

ISO/IEC defines security in [6] as follows: The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them. Furthermore, in some sources security is thought to be a composition of confidentiality, integrity and availability [7, 8]. Common criteria [8] list security failures for these attributes as unauthorised disclosure, modification, and loss of use, respectively. [8] defines an asset as an entity that someone presumably places value upon. This means that almost anything can be an asset. However, in this work we define an asset as data stored in a device or a message in a communication channel – as depicted in Fig. 1.

Risk management is the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level [9]. Risk management utilises threat likelihood and the impact caused by a threat exercise to calculate a risk level. The NIST (National Institute of Standards and Technology) integrates risk management to the SDLC (Software Development Life Cycle), i.e. 1) initiation, 2) development or acquisition, 3) implementation, 4) operation or maintenance, and 5) disposal in [9]. Naturally, phase number four – operation or maintenance – can be thought of as a phase where run-time security adaptation is applicable. The NIST 800-30 [9] suggests performing risk management activities in phase four periodically or whenever major changes are made to the system or its environment. However, in smart spaces these changes in the execution environment happen constantly, which we thought of as context changes.

There are several definitions for context, for instance from Chen and Kortz [10]: Context is a set of environmental states and settings that either determines an application's behaviour or in which an application event occurs and is interesting to the user. In this work we view context from two perspectives. Firstly, the environment context means broadly an application's execution environment, i.e. location, devices and people around, execution platform and so on. Secondly, the usage context means the purpose of an application's usage, i.e. entertainment, work and so on.

Fig. 2 presents our vision of run-time security management published in [11]. In this initial vision, service discovery, adaptation, measuring, and reasoning phases form a sequential process with a feedback loop. Adaptation is performed based on available services, their security properties, and the desired security. Next, the achieved security is monitored by means of measuring. The achieved security value acts as an input for the reasoning activity. If the desired security is not achieved, the reasoning phase calls the adaptation and an alternative security mechanism is selected. The context affects the desired and achieved security levels as well as available security mechanisms and resources. Now in this work, we concentrate on the adaptation phase and its prerequisites.

Adaptation is the ability of software to adapt its functionality according to the environment and usage context, refined from [12]. In this work, functionality means those security mechanisms intended to achieve a particular level of security.

In [13] Zhou defines ontology and its possible use as follows: Ontology is a shared knowledge standard or knowledge model defining primitive concepts, relations, rules and their instances which comprise topic knowledge. It can be used for capturing, structuring and enlarging explicit and tacit topic knowledge across people, organizations and computer and software systems. There are several security ontologies available for different purposes, listed and compared, for example in [14]. Our earlier work also compares a few security ontologies from the viewpoint of run-time applicability [11]. The existing security ontologies are intended to be used in design-time – like ontologies in [15, 16] – or alternatively, for service discovery and matchmaking purposes – like those ontologies in [17, 18]. Added to these, Herzog et al. presented a comprehensive ontology of information security in [19].

Based on Savola et al. [20] security metrics are used for decision support, especially in risk management for mitigating, cancelling or neglecting security risks. Therefore, metrics that might be useful for different purposes will be associated with risk analysis. Security metrics and measurements can be used for decision support, especially in assessment and prediction [20]. In this work, security measurements are used to detect the difference between the required and achieved security level of an application.

Figure 1. Assets in our work

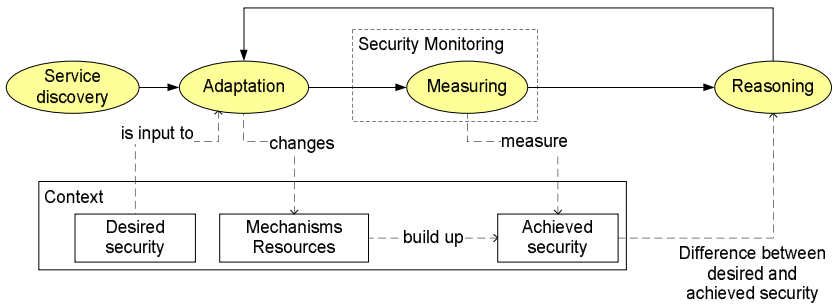


Figure 2. Vision of run-time security management

III. Security Ontology and Measurements

Security adaptation at run-time requires that devices and applications in these devices are able to communicate and exchange security-related information. To make this communication possible security ontologies are needed. Support for run-time security adaptation requires that the security ontology contains security mechanisms, objectives, security measurements, and connections between these. From these concepts, the connections are essential for adaptation, since they make it possible to know which objectives, i.e. high level requirements, can be achieved with a particular mechanism. The security ontology utilised in this work adopts security objectives and mechanisms from [18, 19]. However, measurement part conform the ontology from Savolainen et al. [16]. TABLE I. presents an initial part of a security ontology combined for this work. The first column lists classes, the second column lists the properties of each class and their ranges, and the third column gives examples of possible instances. The ontology definition is work in progress but the current version presents the most important concepts from the run-time adaptation viewpoint. Ontologies in [18, 19] contain dozens of objectives and security mechanisms – all of these are important but copying them to this paper is not reasonable at this point.

Some classes of the security ontology are quite straightforward – listing mechanisms etc. Whereas, some other classes are more complex, like classes related to security measurements. For this security ontology, we defined risk level measurements for confidentiality and integrity based on the proposal in [21] presented in TABLE I. which conforms to the idea of a risk-level matrix from [9]. The vertical axis denotes a threat's likelihood and the horizontal axis means the severity of consequences. We posit that the severity of consequences is an analogue for an asset's value defined in the security ontology – in NIST 800-30 this is defined as impact [9].

For defining likelihoods of threats we have to know what these threats are, and thus we decided to use the following high level threats for confidentiality and integrity respectively: "unauthorised disclosure of information" and "unauthorised modification of information" defined in [8]. Based on these threat definitions and the table above we can define separated risk measurements for confidentiality and integrity as follows:

$$RiskForConfidentiality = Tl * Av \quad (1)$$

$$RiskForIntegrity = Tl * Av \quad (2)$$

Where *Tl* means threat's likelihood and *Av* is an asset's value. Hence, values for these risk levels are calculated in a similar way as TABLE I. shows but it is notable that the values are in the interval [0, 100] not only as values presented in the table. However, we map these numerical values to the risk levels – similarly as made in [21], i.e. low (0-10), moderate (11-39), and high (40-100).

TABLE I. Part of Security Ontology

Class	Properties {Range}	Possible instances
SecurityAsset	hasValue {[0, 100]}	Message Data
SecurityObjective	achievableWith {securityConcept} fulfilmentMeasured-With {SecurityMeasurement}	Confidentiality Integrity Availability Authentication
SecurityConcept	support {securityObjective} utilises {credential}	AES and DES support Confidentiality MD5 supports Integrity
Credential		Fingerprint, Password
SecurityMeasurement	hasValue measuredObjective {securityObjective}	RiskForConfidentiality RiskForIntegrity
Device	hasSecurityConcept {securityConcept} hasCurrentlyUsed SecurityConcept {SecurityConcept} hasMeasuredSecurity {SecurityMeasurement}	MobileDevice with entertainment application

TABLE II. RISK LEVELS ADOPTED FROM [18]

Threat Likelihood	Severity of consequences			
	Insignificant (10)	**Marginal (40)**	**Critical (70)**	**Catastrophic (100)**
Incredible (0.1) Level 1	10 *0.1=1	40*0.1=4	70*0.1=7	100*0.1=10
Improbable (0.2) Level 2	10 *0.2=2	40*0.2=8	70*0.2=14	100*0.2=20
Remote (0.4) Level 3	10 *0.4=4	40*0.4=16	70*0.4=28	100*0.4=40
Occasional (.0.6) Level 4	10 *0.6=6	40*0.6=24	70*0.6=42	100*0.6=60
Probable (0.8) Level 5	10 *0.8=8	40*0.8=32	70*0.8=56	100*0.8=80
Frequent (1.0) Level 6	10 *1.0=10	40*1=40	70*1=70	100*1=100

As said in section 2, an asset can be almost anything but in this work an asset is defined as a message in a communication channel or data stored in a device. However, an asset's value (*Av*) for its owner depends on the content and usage of the asset. Hence, we do not try to define these asset values in this work – instead this information will be collected from user preferences. For calculating threat likelihood *Tl* we propose the following equation:

$$\begin{cases} Tl = 0, \text{ when } Da \geq Ia \\ Tl = Ia - Da, \text{ when } Da < Ia \end{cases} \quad (3)$$

Ia means actions that increase the threat likelihood and *Da* actions that decrease the likelihood. At the moment, we do not try to give numeric values for *Da* and *Ia* variables – instead variables get values describing how many levels they affect to the threat likelihood. As TABLE I. shows *Tl* gets values from the interval [0, 1], and these values are mapped to six levels from incredible (level 1) to frequent (level 6). Naturally, when *Da* is bigger or equals with *Ia* then *Tl* gets value 0, which maps to the incredible level. In this time, we make an assumption that *Ia* and *Da* are independent from each other, e.g. change in a *Da* value does not affect to an *Ia* value. In order to achieve automatic adaptation, calculation of this formula has to be made automatically during the execution based on information stored in the ontology and context information collected from the environment. It is important to bear in mind that we can give exact definitions for *Tl* values 0 and 1 as Hunstad et al. did in their work [22] for probabilistic security values, but precise values between these end points are more complicated as they mention. Hence, we know that threat likelihood 0 means that there is no possibility for threat's realisation, and likelihood 1 means that the threat will certainly be realised.

Utilisation of security mechanisms affects the variable *Da* (Decreasing actions). Therefore, the security ontology has to contain a connection from mechanisms to *Da* variable in the measurement class. For instance, measurement *RiskForConfidentiality* is affected by encryption algorithms like AES (Advanced Encryption Standard) and DES (Data Encryption Standard). Thus, the device software has to select the appropriate encryption mechanism and its parameters to decrease the *RiskForConfidentiality* measurement's value. TABLE III. contains security mechanism assessment, i.e. how many levels mechanisms reduce threat likelihood. Humstad et al. [22] note that the rating of security functions is a difficult task. Thus, it is impossible to give exact values how much a particular security mechanism decreases the particular threat. However, we defined initial *Da* values for security mechanisms in the security ontology in order to test how well the security ontology is able to support run-time security adaptation. The current usage of security mechanisms can give suggestive values for *Da* value selection, for example Anderson mentions in [23] that NSA (National Security Agency) approved AES with 128 bit keys for secret purposes and AES with 256 bit keys for top secret purposes. However, values in TABLE III. change when time goes on because vulnerabilities will be found and attackers learn new ways to attack. In addition, use cases in different contexts will offer valuable feedback for assessing these values. Thus, values in the ontology have to be updated when new knowledge appears.

On the contrary, environment issues affect the variable *Ia* (Increasing actions). Basically these can be anything that happens in the application's environment, i.e. knowing the context is essential to elicit this information. Added to this, the following information can be also used when estimating the increasing actions variable (*Ia*): asset's value because it may be a motive for an attacker, information from vulnerability databases, and log information from intrusion detection systems (IDS). In any case, it is better to overestimate this variable. 0gives examples of events that increase threat likelihood *Tl*.

As a conclusion the security ontology describes security objectives, mechanisms supporting these objectives, and risk based measurements for measuring fulfilment of these objectives. Therefore, the security ontology contains essential information for application adaptation – presented in the next section.

TABLE III. INITIAL SECURITY MECHANISM ASSESSMENT

Security mechanism / parameters for confidentiality	**Initial *Da* value**
AES 256 bit	Reduce 5 Levels
AES 192 bit	Reduce 4 levels
AES 128 bit	Reduce 3 levels
DES	Reduce 1 level
Security mechanism for integrity	**Initial Da value**
SHA-2	Reduce 4 levels
SHA-1	Reduce 3 levels
MD5	Reduce 2 levels
MD4	Reduce 1 level

TABLE IV. INCREASING ACTIONS

Event in the environment	Initial Ia value
Untrusted agent in the environment	Increase 2 levels
Abnormal behaviour of agent	Increase 3 levels

IV. APPLICATION ADAPTATION

The adaptation takes place at the start-up and run-time phases alike. Clearly, the start-up phase adaptation is performed when an application is started. The run-time phase adaptation takes place during application execution – when a major change occurs in the environment or usage of an application. In practice, both adaptation cases are similar from the adaptation viewpoint because similar input information collection and mechanism selection activities are required in both cases. However, the adaptation during the run-time phase requires more sophisticated mechanisms from an implementation perspective, e.g. re-establishing network connections, when compared to the adaptation at start-up phase. In our adaptation approach, the above described security ontology and security measurements are used in both start-up and run-time adaptation phases. The approach is initially intended for smart spaces but the purpose is that it can also be utilised in other environments. The block diagram of the application's adaptation is presented in Fig. 3 – containing both start-up and run-time phase adaptations. The diagram shows the actions required to perform before or during the action of adaptation. In the figure, rectangles mean information and rounded rectangles depict actions. Grey boxes represent information which is applicable only during the run-time phase. Furthermore, in the start-up phase we use the term collecting input information instead of the monitoring term, used at the run-time phase.

A. Start-up Phase Adaptation

The first action is elicitation of security requirements and the required levels for these requirements. The context of application affects these, especially the context related to the usage of application. As an example, entertainment usage and surveillance usage of an application elicit different security requirements and related levels. For entertainment usage, only availability matters and its required level is quite low. This is contrary to surveillance usage which requires availability and integrity at a critical level – adopting terms from TABLE I. Thus, an application gets a list of security requirements and levels. After that, security mechanisms supporting these requirements can be retrieved from the security ontology. Retrieving supporting mechanisms from the security ontology is possible because ontology contains connection from objectives to mechanisms, i.e. *achievableWith* property presented in TABLE I. For instance, if confidentiality and integrity is required retrieving ontology produces a list of supporting mechanisms – like AES, DES, and MD5.

Next, a control analysis action is performed. This is required because the security ontology is intended to be general, and thus may contain mechanisms not implemented in an application or alternatively not applicable in the current environment. The outcome from the control analysis action is a set of applicable security mechanisms. The final action in the collecting input information activity is measuring. In this action, the forthcoming security of the application is measured, meaning the security level that will be achieved without any security mechanisms or alternatively with default mechanisms defined in user preferences when the application starts.

As a conclusion, we have the following information available for the adaptation: security requirements and levels, applicable security mechanisms, and measured security levels. The adaptation action is executed if the measured security fails to reach the requirements set. Based on input information, the adaptation action selects the security mechanisms to be used and their parameters.

B. Run-time Phase Adaptation

The run-time adaptation phase is for the most part similar to the above described start-up phase adaptation. However, some differences exist. Firstly, the adaptation action is triggered based on monitoring results. Thus, the adaptation is performed when required – not only periodically at a moment defined earlier. Secondly, the manner in which adaptation takes place differs when the application is already running, for instance re-establishing network connection might be needed. The monitoring activity monitors earlier mentioned context changes, i.e. changes in usage context and environment context.

The change in the usage of an application may cause change in security requirements or their levels. Consequently, the currently used security mechanisms do not fulfil these new security requirements – recognised by means of measuring. The security ontology defines connection from security objectives to security measurements, i.e. *fulfilmentMeasuredWith* property presented in TABLE I. Thus, ontology offers knowledge needed to measure requirements fulfilment.

Another possibility for changes comes from the environment. These changes affect how well the required securities are met – not the requirements themselves. For instance, a new device appears that poses a threat to the application's operation, or alternatively, the environment changes in a way that prevents the usage of the earlier selected security mechanism. Both of these context changes can create a need to adapt security mechanisms or the parameters of the application.

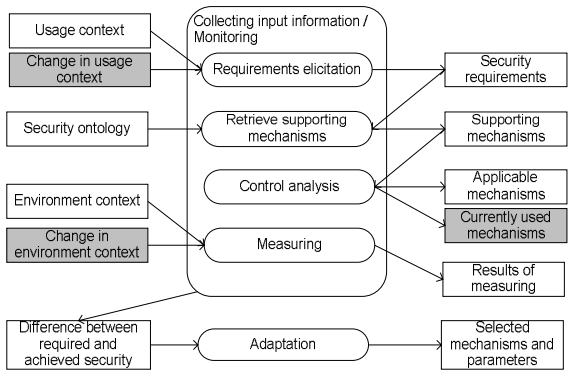


Figure 3. Adaptation block diagram

V. CASE EXAMPLE

We tested our run-time adaptation with a greenhouse demonstration [24]. The demonstrator does not cover the whole adaptation presented in the previous section, i.e. automatic requirements elicitation is not yet implemented – which is required when usage context changes. However, start-up and run-time adaptation is built with the above described ontology and measurements, and changes in the environment context. Therefore, demonstrator measures requirements fulfilment and adapts application accordingly. The smart greenhouse demonstrator contains a miniature greenhouse with several sensors and actuators. In addition, the demonstrator contains two stakeholder groups, i.e. a gardener and the customers of the greenhouse. The run-time adaptation is performed in the gardener application, running in a mobile device. Hence, gardener's device uses security mechanisms only when needed, and thus, is able to save resources.

The demonstrator is built upon the Smart-M3 platform [25] – intended to establish smart spaces where devices can use information and its semantics. The main concepts in the Smart-M3 are SIB (Semantic Information Broker) and agents. SIB mediates information between agents which produce and consume information. In other words, agents do not communicate directly with each other. In the demonstrator, SIB is running in a Linux laptop offering a wireless TCP connection for agents.

A. Start-up Phase Adaptation

Firstly, a gardener arrives at the greenhouse, which is his place of work. The greenhouse smart space authenticates the gardener and assigns him worker privileges. Thus, the gardener is able to look at sensor values from the greenhouse and to change actuator states. The gardener performs these actions with his mobile device, Nokia N810 Internet tablet, containing the gardener's application implemented by Python – user interface is presented in Fig. 4. The first security adaptation takes place immediately when the gardener receives his worker privileges and joins the greenhouse smart space, i.e. security adaptation at start-up phase. The adaptation goes as presented in Fig. 3 – thus the required securities and levels are first decided. The application is intended for working purposes. Therefore, both confidentiality and integrity are critical security requirements and related risk levels have to be in the low level. For these requirements, the security ontology returns the following list of mechanisms for the gardener's application: AES or DES for confidentiality and MD5 for integrity. However, control analysis reveals that the gardener's application does not support DES encryption, and thus the applicable mechanisms are AES and MD5.

The final thing before adaptation is security measuring – based on measurements presented in section 3. This moment, the greenhouse smart space contains only trusted agents, i.e. agents for sensors and actuators. Thus, threat likelihood variable *Tl* – related to confidentiality – in equation 3 remains in level 1 (incredible) because environment does not contain any increasing actions (*Ia*). However, the consequences of possible security breaches might be remarkable harmful to the gardener or the greenhouse, and thus, the severity of consequences (Asset Value *Av*) is in a critical level, when compared to TABLE I. With these values we get following *RiskForConfidentiality* measurement, as TABLE I. shows:

*Incredible (0.1) * Critical (70) = Low level (7)*

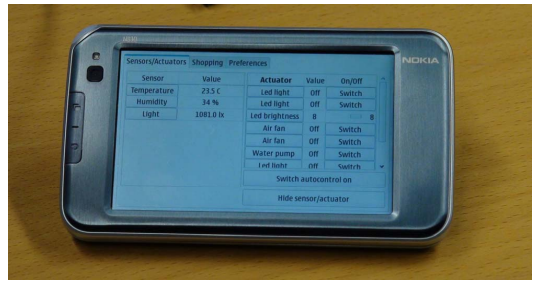


Figure 4. Gardener application in Nokia N810

Similarly a value for *RiskForIntegrity* measurement can be calculated by utilising the same asset value, i.e. critical. However, the threat likelihood (*Tl*) value for integrity risks differs from confidentiality because there has been a possibility that some untrusted third party device has manipulated sensor values before the gardener arrived. In 0it is defined that untrusted agent in the environment increases *Ia* value with 2 levels. Thus, *Tl* increases from the first level to the third level, i.e. to remote level. Hence, utilising equation 2 *RiskForIntegrity* measurement gets value 28 as follows.

*Remote (0.4) * Critical (70) = Moderate level (28)*

Based on these calculations, the gardener's application makes a decision to use hash function (MD5) in communication to ensure integrity, but without encryption because the *RiskForConfidentiality* was below 10, i.e. in the low level. Utilisation of MD5 hash function decreases *Tl* value to level 1 and ensures that *RiskForIntegrity* value also decrease to the low level, which was required. The upper sequence diagram in Fig. 5 shows these start-up adaptation actions.

B. Run-time Phase Adaptation

In the second phase, the retail area of the greenhouse opens and customers arrive to the greenhouse, causing a need for run-time adaptation. The monitoring action recognises that the environment context changes due to these customers and their mobile devices joining to the greenhouse smart space. The required security does not change because the context change comes from environment. However, the arriving customers cause the value of the *RiskForConfidentiality* measurement to increase. This is because customers' mobile devices are regarded as untrusted agents, i.e., an increasing action (*Ia*) in equation 3. *Tl* value increases from the first level to the third level and asset value *Av* remains at the critical level – as described earlier. Using equation 1 *RiskForConfidentiality* measurement reveals that risk level is raised to the moderate level. Hence, the required confidentiality level is no longer reached – causing a need to run-time phase adaptation.

Next, retrieval of supporting mechanisms and control analysis actions are performed. It is also possible to use the security mechanisms list retrieved earlier, in the start-up phase, but it may contain mechanisms already cracked. In this phase, the control analysis action returns two separated lists, i.e. applicable mechanisms (AES) and currently used mechanisms (MD5). Based on this, AES encryption with a key length of 128 bits is selected for ensuring confidentiality. TABLE III. showed that this mechanism decreases *Tl* value three levels, which is enough to reaching risk level low. The lower sequence diagram in Fig. 5 shows the run-time phase adaptation.

The case example showed that it is possible to adapt an application both at the start-up and run-time phases. The security monitoring is able to detect changes in the current security level by means of measuring. Thus, adaptation is performed when required, which ensures selection of the optimal security mechanism.

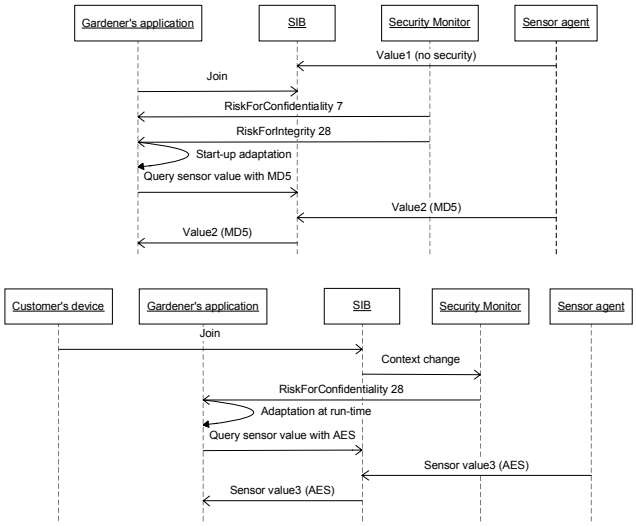


Figure 5. Sequence diagrams for start-up phase and run-time pahse adaptation.

VI. Discussion and Future Work

It is possible to find some essential differences when comparing our approach to the related work mentioned in the introduction section. There are, however, also a few similarities. The Security Adaptation Management (SAM) [1] adapts the used implementation if the current one does not fulfil the required security. However, the adaptation is intended to take place in attack and buffer overflow situations. In some cases, recognising an attack is difficult, or alternatively, when an attack is recognised it might be too late to adapt the software. Our approach recognises security related risks that might cause security breaches, which makes it possible to operate proactively. SAM uses adaptation spaces for defining security breaches and their countermeasures. In our work, this information is stored in the security ontology by using security requirements and mechanisms. The adaptation model presented in [2] concentrates strongly on trade-offs. The presented model makes it possible to select an application variant that best satisfies the user preferences in the current context. In this adaptation model, the user gives the required securities and their importance levels in user preferences. By contrast, our approach elicits these requirements from context information. The adaptive SSL presented by Lamprecht et al. [3, 4] adapts security mechanisms of the SSL session. Performance is a trigger for the adaptation. Authors mention a possibility to use a threat level as a trigger for adaptation but this area is not covered yet. Naturally, the adaptive SSL concentrates to adapt security mechanisms of communication. Thus, it is not able to adapt security mechanisms used to protect assets in a user device. Finally, when compared to design time security variability approaches like [5] it can be noted that the run-time phase adaptation offers more flexibility, especially in those cases where all security related requirements cannot be defined at design-time. As a conclusion, all of these approaches are intended to select the most suitable security mechanism to fulfil security requirements in different situations. Thus, the most significant difference comes from the techniques used to trigger these adaptations.

Risk based security measurements ensure that our approach is able to work in different threat situations. In addition, it is easy to add new security measurements to the ontology. Moreover, current measurements take an asset's value into account. As an example – from our case study domain – data from a temperature sensor is not valuable for a customer but might be very important for the gardener who is making decisions based on this sensor data. Thus, an asset's value is thought to be an important factor in adaptation. The adaptation utilises context information to observe changes in environment and usage of the application. At this moment, changes in the environment context are monitored. However, in the future we will also monitor the usage context of an application that is used to elicit security requirements. The security ontology used in this work is under construction. The ontology is combined from three separate sources containing essential parts for the adaptation, such parts as measurements, mechanisms and supported security objectives. Nevertheless, additions and refinements are still needed, especially related to measurements, context issues, and connections between different concepts. Future studies will concentrate on security measuring and context issues and how different contexts affect the required and achieved securities.

Additional research is also needed in security measurements and their calibration techniques. Measurements related to security mechanisms also facilitate evaluation of how good a particular mechanism really is – because in this work we were obliged to use estimated values in calculations. Moreover, it is important to make performance tests, in order to see how much memory and CPU consumption our approach demands. However, the performed case example does not reveal any major overheads. Also, negotiation techniques between devices should be taken into account because security mechanisms used in a communication requires that both sides support a selected technique. In our case example gardener's application and SIB supported same security mechanisms but sometimes more sophisticated negotiation techniques will be needed. Applicable solutions for the negotiations can be found for instance from existing service matchmaking approaches.

The presented case example is based on the first laboratory experience. The case example is intended to give an overview of the applicability of the adaptation approach and facilitate the future field tests. Currently, we are implementing the wider demonstrator. The second demonstrator contains also requirements decision part based on the usage context of the application. In addition, the second demonstrator utilises more sophisticated security measures as a trigger for an adaptation action. Therefore, the results from the second demonstrator offer valuable input for the evaluations related to feasibility of adaptation decisions and performance overhead.

VII. Conclusions

In this work we presented our run-time security adaptation approach that utilises security ontology and measurements. The high level risk measurements for confidentiality and integrity are presented. Adaptation takes place either at an application's start-up or during the run-time phase. In addition, adaptation is performed at the mechanism and parameter levels, which is essential for many security mechanisms. The designed security ontology supports the adaptation approach and offers a possibility for future changes and extensions. The case example showed that software is able to adapt its security mechanisms during run-time even in a mobile device.

Acknowledgment

This work has been carried out in the ARTEMIS JU SOFIA project funded by Tekes, VTT, and the European Commission.

REFERENCES

[1] H. Hinton, C. Cowan, L. Delcambre and S. Bowers. "SAM: Security Adaptation Manager," *Proceedings of 15th Annual Computer Security Applications Conference 1999 (ACSAC '99),* pp. 361-370, 1999.

[2] M. Alia and M. Lacoste. "A QoS and security adaptation model for autonomic pervasive systems," *32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008,* pp. 943-948, 2008.

[3] C. J. Lamprecht and A. P. A. van Moorsel. "Runtime Security Adaptation Using Adaptive SSL," *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium,* pp. 305-312, 2008.

[4] C. J. Lamprecht and A. P. A. van Moorsel. "Adaptive SSL: Design, Implementation and Overhead Analysis," *First International Conference on Self-Adaptive and Self-Organizing Systems, 2007. SASO '07.,* pp. 289-294, 2007.

[5] V. Myllärniemi, M. Raatikainen and T. Männistö. "KumbangSec: An approach for modelling functional and security variability in software architectures," *First International Workshop on Variability Modelling of Software-Intensive Systems,* pp. 61-70, 2007.

[6] ISO/IEC 9126-1:2001. *Software Engineering - Product Quality - Part 1: Quality Model.* 2001,

[7] A. Avižienis, J. -. Laprie, B. Randell and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, pp. 11-33, 2004.

[8] ISO/IEC 15408-1:2009, *Common Criteria for Information Technology Security Evaluation - Part 1: Introduction and General Model.* International Organization of Standardization, 2009,

[9] G. Stoneburner, A. Goguen and A. Feringa. "Risk management guide for information technology systems," *Special Publication 800-30,* 2002.

[10] G. Chen and D. Kotz. "A Survey of Context-Aware Mobile Computing Research," *Technical Report TR2000-381*2000.

[11] A. Evesti, E. Ovaska and R. Savola, "From security modelling to run-time security monitoring," *European Workshop on Security in Model Driven Architecture (SECMDA),* pp. 33-41, 23 - 26 Jun. 2009. 2009.

[12] M. Matinlassi and E. Niemelä. "The impact of maintainability on component-based software systems," *29th Euromicro Conference,* pp. 25-32, 2003.

[13] J. Zhou. "Knowledge Dichotomy and Semantic Knowledge Management," *Industrial Applications of Semantic Web,* pp. 305-316, 2005.

[14] C. Blanco, J. Lasheras, R. Valencia-García, E. Fernández-Medina, A. Toval and M. Piattini. "A systematic review and comparison of security ontologies," *3rd International Conference on Availability, Security, and Reliability (ARES 2008),* pp. 813-820, 2008.

[15] B. Tsoumas and D. Gritzalis. "Towards an Ontology-based Security Management," *20th Advanced Information Networking and Applications 2006 (AINA 2006),* pp. 985-992, 2006.

[16] P. Savolainen, E. Niemelä and R. Savola. "A taxonomy of information security for service centric systems," *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007),* pp. 5-12, 2007.

[17] G. Denker, L. Kagal and T. Finin. "Security in the Semantic Web using OWL," *Information Security Technical Report 10(1),* pp. 51-58. 2005.

[18] A. Kim, J. Luo and M. Kang. "Security Ontology for annotating resources," *LNCS,* vol. 3761, pp. 1483-1499, 2005.

[19] A. Herzog, N. Shahmehri and C. Duma. "An ontology of information security," *Techniques and Applications for Advanced Information Privacy and Security: Emerging Organizational, Ethical, and Human Issues* pp. 278-301. 2009.

[20] R. M. Savola and H. Abie. "On-Line and off-line security measurement framework for mobile ad hoc networks," *Journal of Networks, 4(7),* pp. 565-579, 2009.

[21] D. S. Herrmann. *Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI.* 2007,

[22] A. Hunstad, J. Hallberg and R. Andersson. "Measuring IT security - A method based on Common Criteria's security functional requirements," *5th Annual IEEE System, Man and Cybernetics Information Assurance Workshop (SMC),* pp. 226-233, 2004.

[23] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* (2nd ed.) 2008,

[24] A. Evesti, M. Eteläperä, J. Kiljander, J. Kuusijärvi, A. Purhonen and S. Stenudd, "Semantic Information Interoperability in Smart Spaces," *8th International Conference on Mobile and Ubiquitous Multimedia (MUM'09),* 22 - 25 Nov. 2009.

[25] Smart-M3, http://sourceforge.net/projects/smart-m3/

PUBLICATION IV

Knowledge based quality-driven architecture design and evaluation

In: Journal of Information and Software Technology, Vol. 52, No. 6, pp. 577–601.

Information and Software Technology 52 (2010) 577–601

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Knowledge based quality-driven architecture design and evaluation

Eila Ovaska *, Antti Evesti, Katja Henttonen, Marko Palviainen, Pekka Aho

VTT Technical Research Centre of Finland, Kaitoväylä 1, 90570 Oulu, Finland

ARTICLE INFO

Article history:
Received 3 July 2009
Received in revised form 16 November 2009
Accepted 21 November 2009
Available online 6 December 2009

Keywords:
Quality attribute
Model-driven development
Software architecture
Ontology
Evaluation
Tool

ABSTRACT

Modelling and evaluating quality properties of software is of high importance, especially when our every day life depends on the quality of services produced by systems and devices embedded into our surroundings. This paper contributes to the body of research in quality and model driven software engineering. It does so by introducing; (1) a quality aware software architecting approach and (2) a supporting tool chain. The novel approach with supporting tools enables the systematic development of high quality software by merging benefits of knowledge modelling and management, and model driven architecture design enhanced with domain-specific quality attributes. The whole design flow of software engineering is semi-automatic; specifying quality requirements, transforming quality requirements to architecture design, representing quality properties in architectural models, predicting quality fulfilment from architectural models, and finally, measuring quality aspects from implemented source code. The semi-automatic design flow is exemplified by the ongoing development of a secure middleware for peer-to-peer embedded systems.

1. Introduction

Systems and devices embedded into our everyday life are software intensive systems that embody service orientation and produce software services upon which our quality of life, e.g. security and safety depend. Therefore, it is extremely important that manufacturers and service providers who produce software based solutions take care that their development processes are properly facilitated to produce services of high quality. Quality is a term with multi-dimensional meaning, which is fully understood only when its context has been specified. Quality of Service (QoS) has a traditional meaning as a property of communication technologies, i.e. throughput, latency, jitter, error rate, availability, and network security as its sub-characteristics. In the context of service-oriented architectures QoS is defined as dependability (including availability, reliability, security, and safety), maintainability, usability and scalability [60]. From the end-user point of view, QoS is the degree to which an executed service meets the quality requirements set by a user. Thus, QoS quantifies service fitness based on the collective behaviour of a composite of services [48]. From service development point of view, QoS defines a set of quality attributes that a particular service has to fulfil. Thus, quality attributes (QAs) defined in the QoS specification of a service system have to be dealt in each software engineering phase; in requirements specification, architecture design and implementation.

When talking about quality, one needs to define what quality is and why it is required. 'What' gives the explicit definition of required quality properties. 'Why' specifies the context where the quality property definition is understandable and complete. For example, when a customised product is designed, the context can be clearly specified because the quality requirements of all stakeholders are defined. The situation is different as the scope is broad. For example, in the case of a cross-domain architecture only those quality requirements that are common for all domains can be defined explicitly. However, extensive reuse of models and getting remarkable cost savings are possible only if quality requirements have been explicitly defined and represented in models.

There are three main approaches for handling quality properties in designs.

Ontology orientation focuses on representing, structuring and managing topic knowledge shared across people, organizations, computers and software [17]. Several methods for ontology development exist, e.g. METHONTOLOGY [24], and a set of languages can be applied to represent knowledge in a machine readable format, such as eXtensible Markup Language (XML) [9], Resource Definition Framework (RDF) [32], and Web Ontology Language (OWL) [51]. Web Ontology Language for Services (OWL-S) [5] is a specific description language for describing a service ontology.

Software architecture design based on generic modelling technologies addresses Model Driven Architecture [55], Unified Modelling Language (UML) [8] as a common modelling language and dedicated transformation languages for describing transformation rules. Three types of transformation languages are: the Query/

* Corresponding author. Tel.: +358 20 722 111; fax: +358 8 551 2320.
E-mail address: eila.ovaska@vtt.fi (E. Ovaska).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.11.008

View/Transformation (QVT) language [61], XML as an intermediate transformation language and specific action languages.

Domain-specific modelling emphasises the specifics of a domain by defining the primitives of an application domain in the meta-models from which a domain-specific language is derived [49]. Although UML2 is a generic modelling language, it also provides constructs to extend the language with domain-specific concepts. UML with the Modelling and Analysis of Real-time and Embedded systems (MARTE) profile [65] is an example of a generic language adapted to a specific domain.

These approaches have evolved simultaneously but separately. All of these approaches aim to model the concepts and properties in an accurate and reusable way. The main difference is in their focus; knowledge, software, and domain. However, not one of these approaches by itself provides a solution to handle different quality properties systematically from the requirements specification to the architecture design, and finally to the source code. Ontologies support knowledge reuse but not architecture modelling. Moreover, no commonly accepted quality ontologies exist. UML as such lacks of support for modelling quality properties. However, UML can be extended with the MARTE profile [65], a profile that enhance UML2 constructs with the domain-specific quality properties of embedded systems. However, the profile gives only partial support for describing quality properties; performance is covered but, for example, security and reliability are not supported. Furthermore, reuse of modelling constructs of quality properties is not supported in MARTE.

The objective of this paper is to introduce a systematic approach to manage quality properties during the whole life cycle of model-driven development. This paper presents how to manage and trace quality characteristics from requirements specification to architecture design, how designs can be annotated by the predicted and measured quality properties gathered by evaluation and testing, and how these designs can be stored for reuse and shared among developers. Our approach is a fusion of ontology orientation, model driven software engineering, and domain-specific modelling approaches. Our main topic of knowledge is quality and each quality attribute is understood as a separate sub-domain [84]. Ontology orientation is used as a means for dealing with and managing quality knowledge [71]. The model driven approach with UML profiles makes it possible to use quality knowledge and manage its variability at the time of development [57,83]. Separation of quality engineering and software engineering makes models reusable and evolvable.

Our main contribution is a quality aware software architecting approach with supporting methods, techniques, and an integrated tool chain. Some of these methods, techniques, and tools are quality attribute specific; some of them are generic. However, we argue that our approach is generic in the sense that the same principles and the same tool environment with quality attribute specific extensions can be applied to all quality attributes. The common principles form the core of our methodology, and the tool environment is based on an open source platform, Eclipse.[1] To our knowledge, this is the first time that this kind of systematic approach with a supporting tool chain has been introduced.

The main advantages of our approach are; (1) Reusable quality requirements specifications based on a uniform model of quality attributes and their metrics. (2) A systematic transformation of quality requirements to architecture and representing them as quality properties of architectural model elements. (3) Reuse of architectural knowledge, i.e. existing styles, patterns, solutions and test results for architecting and quality evaluation. (4) A tool chain that supports the whole design flow from quality requirements specification to source code testing.

The structure of the paper is as follows: The next section discusses the earlier research results. Section 3 introduces the quality aware model driven approach by discussing the main principles of the approach and justifies the rationale behind them. Section 4 presents a case study introducing how the approach was applied to engineering a novel quality aware middleware for peer-to-peer embedded systems. Section 5 discusses the lessons learned by applying the approach and future research directions. Conclusions close the paper.

2. Definitions and related work

2.1. Multiple views of quality attributes

Quality can be tackled from different angles [18]: (a) Judgmental criteria define quality as the goodness of product. The definition does not provide a means by which quality can be measured or assessed as the basis for decision making. (b) Product quality is a function of a specific, measurable property, and differences in quality reflect the differences in the quantity of some product attribute. (c) User-experienced quality is defined as fitness for intended use, or how well the product performs its intended function. (d) Value-based quality is based on the relationship of usefulness or satisfaction to price. (e) Manufacturing-based quality defines quality as the desirable outcome of engineering and manufacturing practices, or conformance to specifications. Our focus is on product quality and user-experienced quality.

ISO/IEC defines a software quality model [45] according to six categories of quality characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. Quality characteristics are externally or internally observable properties of systems, also called quality attributes (QA). ISO 9126:2-4 [44] define quality attribute metrics for three categories; the *execution* qualities that express themselves in the behaviour of systems and are observable only at run-time; *evolution* qualities which are embodied in the structures of systems and are considered in the development and maintenance of systems; and *quality-in-use* metrics which measure user-experienced quality.

The focal interest of quality attributes for system architectures is in how quality attributes interact with, and constrain, each other, and how they affect the achievement of other quality attributes. That is why a set of quality attributes must be evaluated at the same time and the tradeoffs analysis made for achieving an optimal quality level. For example, dependability is a concept that includes four quality attributes: reliability, availability, safety, and security. Moreover, a new concept 'trustworthiness' focuses on a holistic view of quality including the following attributes: correctness, safety, availability, reliability, performance, security, and privacy. The holistic approach aims at applying multidimensional optimisation techniques on top of a set of quality attributes that can have intrinsic and/or extrinsic relationships on other quality attributes. An intrinsic relationship defines the inherent properties of a quality attribute and their relations. For example, an attack that harms system's availability is dealt as a security threat. Thus, there is an intrinsic relationship between security and availability; an increase in availability increases security. Extrinsic relationships occur when attributes behave in opposition. For example, if an increase in reliability decreases performance, the relation between reliability and performance is extrinsic. However, the relations between two attributes do not exist per se but they are rather properties of system architecture [31].

[1] http://www.eclipse.org.

2.2. Quality attribute ontologies

Ontology is a shared knowledge standard or knowledge model defining primitive concepts, relations, rules and their instances, which comprise topic knowledge [82]. Ontology can be used for capturing, structuring, and enlarging explicit and tacit topic knowledge across people, organizations, and computer and software systems [17]. In this work, we use ontologies to describe and specify quality attributes in a uniform way. Thus, quality ontologies make it possible to communicate effectively between stakeholders relating to quality attributes. Ontological engineering is a branch of knowledge engineering, viewed as a methodology and toolset for developing and managing knowledge ontologies.

Quality attribute (QA) ontologies consist of concepts and relationships for capturing and structuring quality knowledge related to product quality. Quality of Service ontology defines the concepts and relationships related to user-experienced quality. QoS ontology includes user-related concepts and the quality attributes relevant to services with which the QoS ontology is used. Thus, a set of QA ontologies is prerequisite for QoS ontology. However, commonly accepted QA ontologies do not exist.

QoS/QA specification languages constitute a set of qualifiers in using QoS/QA ontology, for example, usage range, domain, and meaning, enabling service customers and providers to unambiguously communicate quality. It is possible to have many QoS specification languages for a single QoS ontology.

Efforts for defining QA ontologies have resulted in different kinds of ontologies; e.g. dependability and security ontologies have been aimed at systems engineering and security service development, respectively. However, security ontologies defined for web service annotation, security mechanisms and information security are applicable only for their own specific purposes. Moreover, most of the metrics are process metrics and are not suitable for measuring product-quality or the fitness of service-oriented systems.

FIPA-QoS ontology contains a basic vocabulary for the Quality of Service. Agents can utilise FIPA-QoS ontology when communicating about Quality of Service. Thus, agents can query current QoS values from another agent, or alternatively, an agent can subscribe a notification when something happens related to the QoS. For instance, the agent can be informed when the throughput value of the sent real-time data drops below the required QoS value [25].

DAML-QoS ontology constructed by Zhou et al. [80] complements DAML-S (nowadays OWL-S) ontology to take also a service's non-functional aspects into account. DAML-QoS ontology contains three layers: (1) the QoS profile layer for matchmaking purposes, (2) the QoS property definition layer for elaborating the property's domain and range constraints, and (3) the metric layer for metrics definition and measurement. DAML-QoS makes web services machine understandable and especially facilitates service discovery. For example, the service enquiry returns services that may potentially satisfy the desired QoS requirements. DAML-QoS provides only integer values for QoS measurements, which are also used in the QoS measurement framework [81].

Tian et al. [74] have defined a WS-QoS approach for dynamic QoS-aware web-service selection and monitoring. One part of the WS-QoS approach is WS-QoSOntology containing definitions for metric, protocol, and priority. However, this ontology is not stored in a standard ontology format (e.g. in OWL (Web Ontology Language) format). Nevertheless, the metric definition part specifies direction for each metric, e.g. higher is better, which is an essential feature for achieving machine reasoning.

Dobson et al. [16] describe a QoS ontology, called QoSOnt, for service-centric systems in order to enable communication about QoS between clients, providers, and intermediaries. The QoSOnt consists of three layers: (1) Base QoS and Units form the lowest layer, (2) Quality attributes constitute the middle layer, and (3) Usage Domains model the highest layer. Each layer is a separate ontology, and third parties can replace these ontologies. The Base QoS ontology defines basic concepts common for all QAs that can be utilised in the higher layers. The Attributes layer specifies a QoS attribute, e.g. for dependability. The dependability ontology may represent dependability attributes – reliability, availability, safety, and security – and a set of metrics for them.

Dependable Systems Ontology [29] also called ReSIST ontology – is an ontology about resilient and dependable systems. The ontology contains two main classes, i.e. Computer System Vulnerability and Dependable Systems Technology. The first class contains threats that can cause a computer system to malfunction or stop – and the latter class contains features and the research areas of dependable systems technology. Mostly, this ontology contains concepts related to dependability and security, but the metrics are not defined.

2.3. Quality attribute modelling

UML [62] is a de facto standard for modelling software architectures. UML2 offers profiles as an extension mechanism for adapting the UML metamodel with constructs that are specific to a particular domain, platform, or method. UML profiles are a natural way to connect the defined quality requirements to the architectural models as stereotypes. Usually UML tools support a set of profiles by default but it is possible to define new domain-specific profiles.

MARTE [65] is a specific UML2 profile for the embedded real-time systems domain. MARTE includes a set of sub-profiles, one of which is the non-functional properties sub-profile, intended for describing quality and other non-functional properties in models. A specific sub-profile for quality evaluation purposes is also provided by the Generic Quantitative Analysis Modelling sub-profile that can be used for schedulability analysis and performance evaluation. As drawbacks, MARTE does not provide a systematic approach for describing all execution qualities, it has no support for reusable quality definitions, it is complicated to use with immature tools and missing guidelines.

Another approach is to extend UML with quality specific ontologies [57]. In this approach each quality attribute forms a quality domain that is represented as a separate QA ontology. W3C has published RDF-S [4] and OWL [51] languages for representing ontologies. The RDF-S is an extension to RDF that makes it possible to represent structures, i.e. classes and sub-classes, in the RDF description. Whereas, OWL is designed for applications that need to process the content of information. OWL contains three upward compatible sub-classes: OWL Lite, OWL DL, and OWL Full.

When QA ontologies are represented by using the ontology languages, they can be merged with application domain-specific ontologies and used as the integrated ontology of a specific domain, e.g. smart houses. Unlike MARTE, this approach manages the evolution of QA ontologies and application specific ontologies separately, i.e. both the quality property models and the architectural models are evolvable. Only mappings between quality property models and architectural models are case-specific. In the use of MARTE, quality property definitions are tightly intertwined with architectural descriptions.

2.4. Model driven architecting and testing

The two major challenges for MDD modelling languages are to raise the abstraction level and to keep formality high enough to support formal manipulation. MDD can be categorised into two schools for tackling the challenge of abstraction: The Extensible General-Purpose Modelling Language School and The Domain-Specific Modelling Language (DSML) School. The first one provides a general-purpose language with domain-specific extensions, e.g.

UML. In the latter, a domain-specific language is defined using meta-modelling mechanisms and tools [26].

MDA is an OMG initiative designed to provide a standardization framework for MDD [55]. The three primary goals of MDA are portability, interoperability, and reusability through architectural separation of concerns. One of its key tenets is direct representation [7]. Direct representation means using models for representing problems rather than using models as graphical syntax for programming languages. MDA specifies three viewpoints on a system; a computation independent viewpoint (CIM), a platform independent viewpoint (PIM) and a platform specific viewpoint (PSM) [55]. Models in the context of the MDA Foundation Model [63] are instances of Meta Object Facility (MOF) meta-models, and therefore, consist of model elements and links between them. That is, meta-models in the context of MDA are expressed using MOF.

MDA comprises a set of non-proprietary standards that will specify interoperable technologies with which to realise model-driven development with automated transformations [72]. However, not all of these technologies will directly concern the transformations involved in MDA. Transformation in the context of MDA means transforming any MOF compatible model to another MOF compatible model. A common transformation language, MOF QVT Specification 1.0, was published in April 2008 [61].

Although being the de facto standard and following the recommendation by OMG that the modelling language used for MDA framework is UML, the alternative is to define DSML. However, according to the MDA specification, the modelling language must be MOF compliant [52].

Model-Based Testing (MBT) facilitates software testing by automating test suite generation and test execution tasks [23]. The MBT is a black-box approach, applied to binary programs without source code, to embedded software, and even to hardware devices [77]. The models used in MBT can focus on modelling the system under test, or the environment of the system (capturing the ways in which the system will be used), or both the system and its environment [77]. The MBT is used to model the particular structural or behavioural aspects of a system for defined objectives, assumptions, and structures [23]. Thus, the model of the system under testing serves two purposes [78]: (1) it acts as an oracle for the system by encoding the intended behaviour and (2) its structure is exploited for generating test cases. An oracle is a component that assigns a pass/fail verdict to each test from the model of a system. The model used in MBT must be simpler than the system, or at least easier to check, modify, and maintain but at the same time the model must be sufficiently precise to serve as a basis for the generation of "meaningful" test cases.

MBT starts typically from abstract visual models that are refined with additional information in order to enable the automatic generation of testing artefacts. Thus, model-based test generation contains usually the following tasks [77]:

Building an abstract model of the system under test, i.e. the model needed for test generation may be a little different to the model needed for other purposes. The abstract model can be just a functional or behavioural model of the system under test, or of the environment of the system, or of both the system and its environment.

Validating the model (typically via animation) for detecting gross errors in the model. If some errors remain in the model, they are very likely to be detected when the generated tests are run against the system under test.

Generating abstract tests from the model. The test engineer can control various parameters to determine which parts of the system are tested, how many tests are generated, which model coverage criteria are used etc.

Refining these abstract tests into concrete executable tests by adding concrete details missing from the abstract model. It is usually performed automatically, after the test engineer specifies a refinement function from the abstract values to some concrete values, and a concrete code template for each abstract operation.

The following lists the benefits of the MBT: (1) The MBT facilitates the creation of behavioural models early in a development life-cycle, thus exposing ambiguities in the specification and design [68]. (2) The MBT supports reuse in testing because in contrast to a test suite, the behaviour models are much easier to update and reuse in any future testing if the software specification changes [68], and finally (3) The MBT potentially supports earlier fault detection and a higher level of coordination between design and testing activities [23].

The *model-driven testing* requires a similar structure to MDA to facilitate, besides the generation of test cases and oracles, the execution of tests on different target platforms [33]. In order to benefit from the separation of PIMs and PSMs in the generation and execution of tests, the strategy of Model-Driven Testing has to refine the classic three tasks of model-based testing of [33]: (1) the generation of test cases from models according to a given coverage criterion, (2) the generation of a test oracle to determine the expected results of a test and (3) the execution of tests in test environments, possibly also generated from models. Tasks 1 and 2 are platform independent tasks whereas task 3 takes place on a certain target platform of the application [33]. Thus, platform specific models are required in task 3 to generate test environments and to map platform independent test cases and oracles on the desired platform. For example, a model transformation technique can be used to generate test cases from a PIM of a software system.

3. Quality aware software architecting

Quality aware software architecting extends model and quality-driven architecture design and evaluation with the means of a knowledge engineering discipline in order to increase the use of existing design knowledge in the development of complex software intensive systems. Although some methods support quality modelling and testing (see Section 2), to our knowledge there are no approach for quality aware software architecting capable of managing both the quality knowledge and the architectural knowledge during the construction of a software system. In order to support quality aware software engineering, we developed a new quality aware architecting approach that incorporates quality knowledge modelling and management and quantitative and qualitative measurement of software quality in the early verification and validation phase of architecture design. Making use of Quality Attribute (QA) ontologies, the supporting tool chain enables software architects and quality attribute experts to formally, explicitly, and coherently conduct architecture modelling and quality attribute modelling in a unified computer-aided environment. As a benefit of our quality aware software architecting approach is the fact that it facilitates software developers to: (1) model reusable quality requirements for software systems, (2) transform the modelled quality requirements to the architectural models, (3) reuse existing design knowledge such as architectural styles, generic design patterns, and domain-specific patterns to achieve desired quality goals in software architecture, and finally (4) evaluate that the desired quality goals are met in the software models and code. This all helps software developers to improve and keep quality of software stable; the systematic approach of quality aware architecting enables to manage both the quality knowledge and the architectural knowledge during the construction of a software system and always use the best knowledge of both design knowledge areas.

Quality aware modelling process (Fig. 1) has three main phases, and two of them that focus on modelling have a knowledge engi-

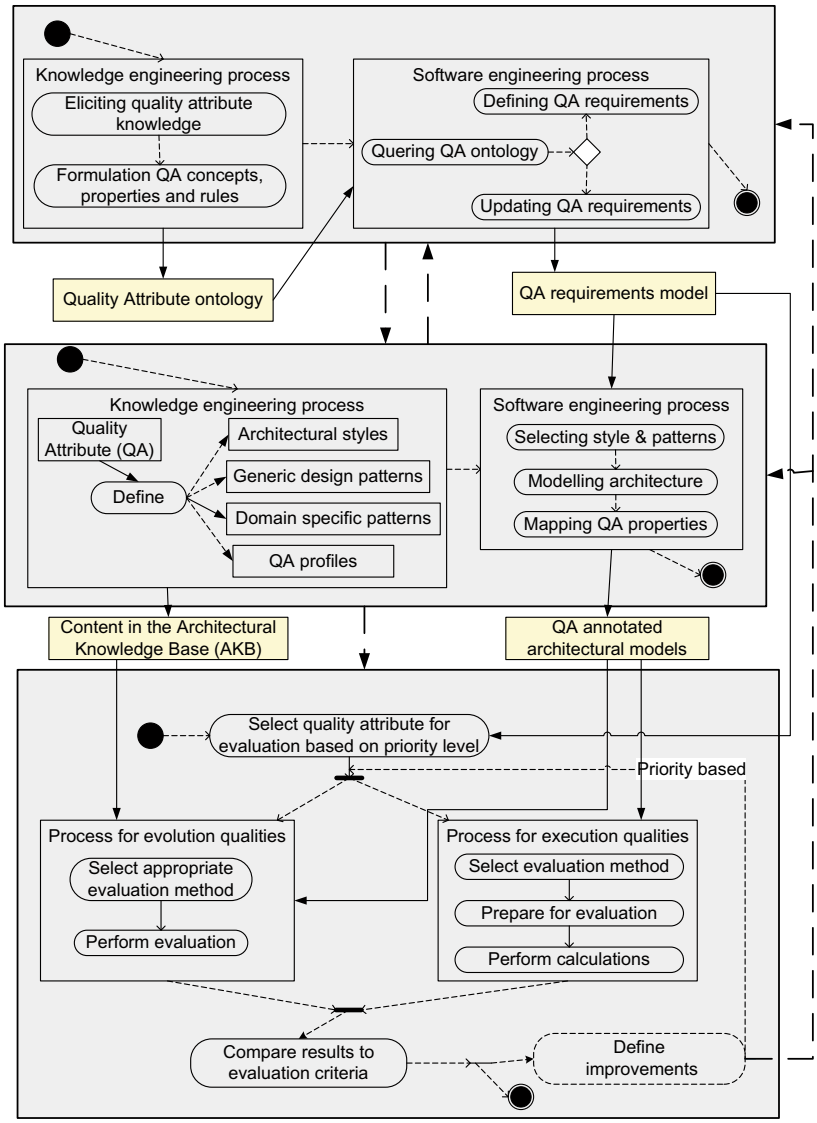


Fig. 1. Quality aware modelling process.

neering process and a software engineering process. Knowledge engineering process focuses on creating and providing knowledge of quality attributes for the use of the corresponding software engineering process. The aim of these two separate processes is to maximise the reuse of the design artefacts produced in the requirements specification, architecture design and quality evaluation phases. Knowledge engineering and software engineering processes have separate activities and the fusion of the results of both processes is made at the information level by design models. In the following sections, the three modelling phases are discussed in more detail phase by phase; modelling quality requirements (Section 3.1), representing quality properties in architectural models (Section 3.2) and evaluating quality fulfilment from models and code (Section 3.3). The whole modelling process is highly iterative. Shortcomings identified in one phase cause re-engineering of the results of the precede phase(s). For example, if required improvements are identified in the evaluation phase, they can cause changes both in quality requirements, and architecture models. Thus, iterations may be needed in one or more design activities.

The quality aware modelling process can be tailored for different software engineering processes, like Rational Unified Process [69] and agile processes such as SCRUM [73]. For instance, our approach covers stakeholders' involvement and all four lifecycles of RUP (i.e. inception, elaboration, construction and transition) to the extent that relates to software architecting. However, our approach emphasises more the activities of elaboration and construction than inception and transition. Due to the iterative and incremental nature of our approach one iteration can include all

design activities; requirements specification, conceptual design, detailed design, quality evaluation, coding and testing. The first two design activities, i.e. requirements specification and conceptual design, are well defined in the same way as in SCRUM. However, our approach differs from agile methods by the way of achieving agility. In SCRUM, agility is achieved by a flexible empirical process that is based on sprint backlogs. Our approach achieves agility by repeating iteration loops of model based design, prioritised evaluation and testing. Therefore, adoption of our approach for SCRUM needs more tailoring than for RUP. Moreover, our process model focuses on technical development of software architecture, not on the management of the whole development work like RUP and SCRUM.

3.1. Modelling quality requirements

The goal of quality requirements modelling (also called QA requirements) is to create two models; the QA ontology and the QA requirements model. The QA ontology captures the knowledge related to a specific quality attribute and is the result of a specific knowledge engineering process. The QA knowledge engineering process has two main phases; (1) QA knowledge is first elicited, and then (2) QA concepts, properties and rules are formulated. This information is presented as a QA ontology.

The QA requirements model, produced by a software engineering process, uses the QA ontology in order to define and update the QA requirements model that is related to certain products or services. As depicted in Fig. 1, both processes result in a model. Thus, the quality attribute related knowledge is separated from the context where the QA knowledge is used. Our goal is to make it easier to manage the evolution of both models; QA ontology represents a commonly accepted understanding of a specific QA domain managed by an organisation or initiative; the QA requirements model represents customers' needs and market relevance and therefore, changes in markets or in customers' needs reflect upon them. By following the '*separation of concerns*' principle, we can make the QA requirements aligned and reusable. QA ontologies make it possible to compare the quality properties of products and services from different suppliers when they apply the standard QA metrics defined by the QA ontology. Thus, QA metrics can be managed and reused among a large set of suppliers. The use of QA ontologies assists in the alignment among company boundaries, whereas the QA requirements model makes the same inside an organisation or/and application domain by improving efficiency of non-functional requirements specification.

3.1.1. Defining QA ontologies

In this section, we focus on the QA ontology definition process presented in the left top corner in Fig. 1. Defining an ontology for a specific quality attribute requires a broad understanding of the quality domain in hand. An adequate expert group may be needed to offer consultation and feedback for the quality engineer. The quality engineer and expert group should develop the QA ontology iteratively together in order to ensure that the needed aspects are taken into account in the QA ontology. Consequently, defining a QA ontology is a time consuming task, and the final result depends on the experience of both the quality engineer(s) and the expert group.

We have defined QA ontologies for reliability and security attributes. These ontologies define the quality attributes in general. The reliability ontology (Fig. 2) was defined based on an extensive literature analysis, including mainly books and standards [83]. Our

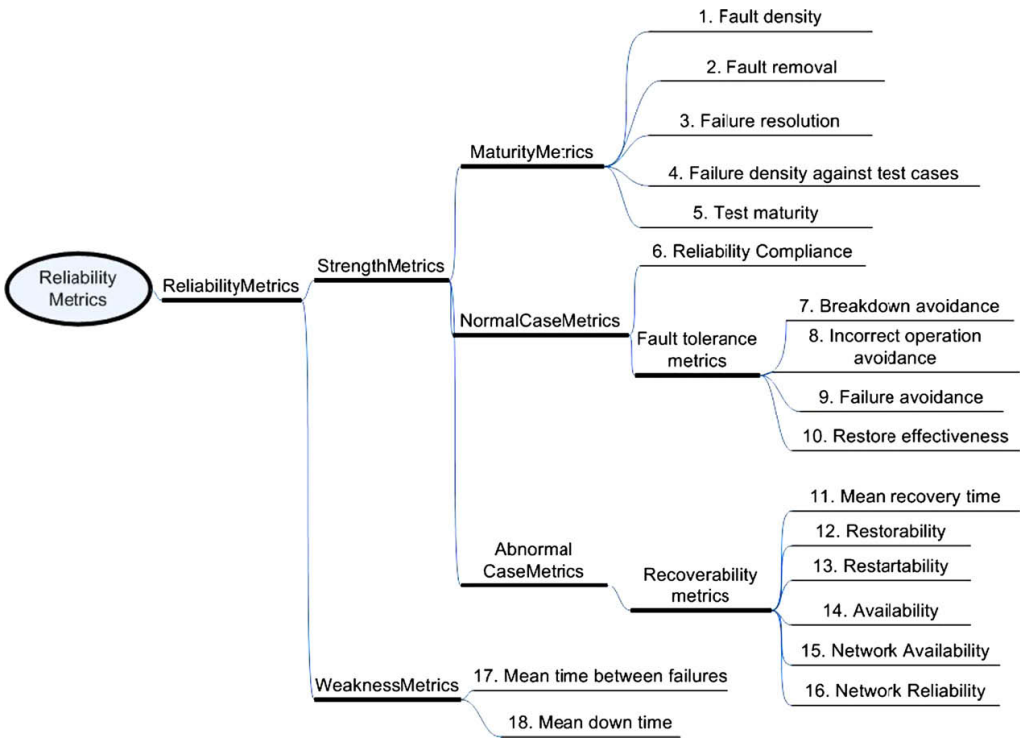


Fig. 2. The reliability metrics [57].

effort on information security covered the analysis of the most relevant literature resulting in security taxonomy (Fig. 3) [71]. Furthermore, our QA ontologies contain *quality metrics* for both setting target values and for measuring fulfilment of the quality attribute requirements. These quality metrics are the results of several iterative design efforts of ours. Both of the defined QA ontologies are aligned and contain a metric class. The different metrics, i.e. instances of the metric class, are represented as individuals under the metric class. Quality metrics use shared properties, which do not change between QA ontologies. In other words, each quality metric has the same properties. These properties are: (1) objective; (2) target, i.e. where the metric can be used; (3) applicability, i.e. when the metric can be used; (4) one or more formulas; (5) a range value for the measurement; and (6) the optimal value of the measurement. Fig. 2 presents the content of the metric class in the reliability ontology as a taxonomy form; a bold line represents a class, and a numbered thin line means a quality metric.

Quality metrics were collected mainly from ISO/IEC 9126 [45] and IEEE 982.1 [38]. Although much effort has been put for quality standardization, software metrics are still an immature area. Especially, execution qualities, such as reliability and security, lack of standard based metrics. Quality standards focus mostly on process metrics. Furthermore, only a few standard based information security metrics were available. Although companies specialised on security have their proprietary solutions and metrics, they are not available, do not relate to information security nor are standard based. The reliability ontology contains 18 metrics (Fig. 2) and the security ontology (Fig. 3) contains 8 metrics. The standard metrics of ISO/IEC 9126 and IEEE 982.1 were classified according to the object of a measurement. It is also possible to extend the metrics classes with new ones if the ontology does not provide suitable metrics. Adaptation by extensions is allowed and needed due to different business and technical goals of companies. Adaptation needs are also obvious because of evolution, i.e. changes in business and technology cause changes in the QA ontologies. However, due to the QA ontologies, changes are manageable and made only in one model. However, if the formula of a metric is changed, it causes changes to existing architectural models and their realisations. Such kind of modification is not tolerated.

The security and reliability ontologies categorise metrics to Strength and Weakness metrics [57,71]. The alignment makes the metrics easier to understand for architects. The separate QA ontologies also makes ontologies more evolvable by enabling different QA engineers (and other stakeholders) to refine ontologies in parallel, necessary if different engineering tasks have to be carried out concurrently in different places. Nevertheless, these separate QA ontologies can be merged if needed. The QA ontologies facilitate knowledge sharing and reuse in the quality requirements specification phase, which many stakeholders are involved in.

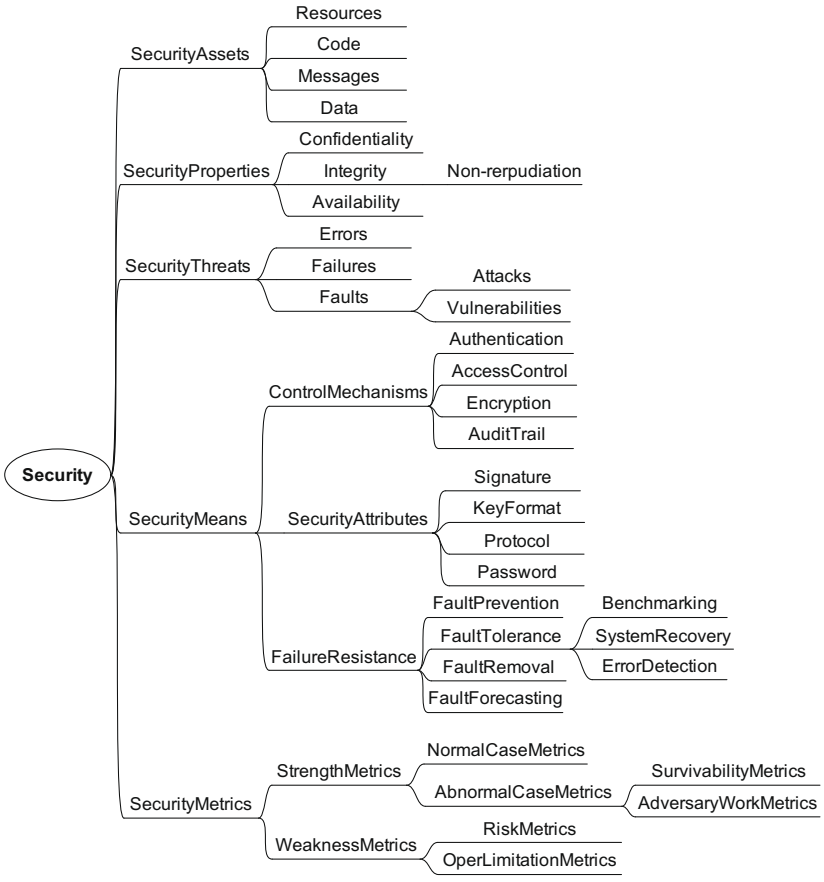


Fig. 3. A taxonomy of the concepts in the security ontology [71].

3.1.2. Defining quality attribute requirements

In this section, we focus on the QA requirements definition process presented in the right top corner in Fig. 1. Specifying quality requirements is a challenging task because different stakeholders typically understand QAs according to different conventions. By contrast, the QA ontology defines the quality attribute in a uniform way; for example, the metric class defines what to measure, where the metric is applicable, how to use the metric, and what formula to use. Due to these uniform descriptions, the software architects can understand the quality attributes in a similar fashion. In addition, the software architect can enhance his/her knowledge related to a particular quality attribute by examining the content of the quality ontology – because the quality ontology contains domain knowledge of a quality attribute. For example, we defined five main concepts for the information security (Fig. 3); assets, characteristics, threats, solutions and metrics [71]. From the security ontology (Fig. 3) the software architect understands that *faults*, *errors*, and *failures* are the main classes of security threats. Furthermore, the architect notices that *control mechanisms*, *control properties*, and *failure resistance* are the main classes of the security solutions. Therefore, the architect can specify appropriate quality requirements for achieving a desirable security level.

The software architect can now derive the quality properties from the requirements specification. The QA ontology helps in this task. For example, secure use of a software system may require usage of both the authentication and the access control mechanisms. Thus, in this case, *use authentication* is the first and *use access control* is the second quality (security) property of the software system. These properties are the sub-classes of the *control mechanisms* class of the security ontology (Fig. 3).

As said in the previous section, defined quality ontologies contain metrics for setting target values and measuring the fulfilment of the quality attribute. In practice, these quality metrics are intended for measuring the fulfilment of the quality property. Therefore, the software architect selects a suitable quality metric for each quality property and sets a desired target value for it. The selection of the quality metrics for the properties is an iterative process in which different stakeholders from business and technology development are involved. However, that process is out of the scope of this paper.

3.2. Representing quality properties in architectural models

Knowledge engineering enriches the quality aware architecture modelling process (middle part in Fig. 1) in four different ways; (1) Architectural styles create the generic basis of architectural knowledge, i.e. styles define a set of types of architectures repeatedly applied in products, upon which architecting is based; (2) Generic design patterns provide solutions applicable across different application domains. (3) Purpose and domain-specific design patterns provide micro models for adjusting architecture models for specific purposes, and finally (4) QA profiles make it possible to tune QA properties further to match the requirements of a specific product or service. The knowledge storage is defined as the Architectural Knowledge Base (AKB). All knowledge in AKB is organised in a way that the styles and patterns are searchable by the name of the quality attributes they support. Thus, architecting is quality driven, not functionality driven. The main reason is that quality requirements influence architecting more than functional ones; quality requirements define which style to be used as a starting point, which patterns to use for refining the selected style(s), where to use the selected patterns, why to use them and not others, and which part of architecture need tuning with the QA profiles. Functional requirements can be achieved by different means but in order to fulfil quality requirements only a few or one option(s) are/is possible. AKB is a repository for reusable models; thus, reuse is made possible among artefacts of the highest abstraction level, where it is the most efficient.

AKB provides assistance and guidance for software engineering process. A tool for searching suitable models (i.e. styles, patterns, reference models, and domain models) for a particular case in hand assists in architecture modelling. The tool enables mapping of QA properties to model elements of the architecture, too. Because, the use of AKB is case sensitive, both of these tools are semi-automatic. Because the architect has the most relevant information of the system to be modelled, (s)he has all rights for decision making, assisted by the AKB and supporting tools.

3.2.1. Quality driven model selection

Quality-driven architecture design relies on the assumption that evolution qualities are embodied in the architectural structures of software. Architectural styles and patterns, and also design patterns, promote different quality attributes. When patterns are applied in the architecture, quality characteristics of the selected patterns are reflected to the entire software architecture [50].

AKB provides uniform information about architectural reference models (e.g. styles and patterns) and their quality characteristics [50]. The AKB helps software architects to select the most suitable styles and patterns for achieving the desired quality goals in a software system [54]. The idea is to move from the "notion of architectural styles into the ability to reason based on quality attribute specific models" [50] and thereby improve the quality of architecture design. In addition, the AKB promotes knowledge sharing in development teams [58]. The benefits of the knowledge sharing capacity are perhaps most evident in the case of domain-specific solutions, since they are often tacit knowledge inside an organisation [21]. The AKB makes such knowledge explicit and provides means to effectively maintain, share and accumulate it [21]. Architecture development becomes faster and more reliable as there is no need to "reinvent the wheel" [1,50].

The AKB contains the following basic information on each model [53,54]: *name*, *alternative names*, *abstraction level* (conceptual/concrete), *type* (e.g. pattern/style/reference architecture) and *intent*, and a list of *supported quality attributes and their rationales*. In addition, each reference model is associated with a picture and a *guide*, which provides instructions on applying the model (e.g. application areas, implementation guidelines, consequences, common problems and related reference models). The models can be searched from the AKB by any of the mentioned fields.

The AKB is used as a model selection guide when designing a new architecture from scratch. Similarly, it can assist in transforming architectural models from the existing style to another when the quality requirements have changed [54]. The architect searches the AKB according to the desired quality characteristics and selects patterns on that basis. First, an architect would select an overall architectural style for the system and then adjust it to specific purposes with generic and/or domain-specific design patterns. In addition to the quality properties of individual patterns, one also has to consider how different patterns compose with each other [50]. For example, let's assume that *modifiability* and *interoperability* are listed as the primary quality goals of a GUI-intensive software system. An architect searches AKB according to these attributes and limits the search to architectural patterns. The search results show that *Layers*, *Blackboard*, *Presentation-Abstraction Control*, and *Model-View-Controller* patterns support modifiability while the *Broker* pattern supports interoperability. The architect then looks closer at these patterns for more information and makes her decision. (S)he could select, for example, the combination of Model-View-Controller and Broker [11]. Model-View-Controller separates the application logic from the user interface (modifiability) while Broker would deal the transformation of the application interface to the properties of the used terminal (portability) [58].

In real-life, decision paths and tradeoffs are obviously much more complicated than in the simplified example above. However, in any situation, the AKB supports architect's decision making process by providing a large amount on information on both general and domain-specific patterns and styles. The AKB lists a set of styles or/and patterns that embody the quality attribute(s) the architect is interested in. The AKD does not make design decisions. That is the duty of the architect because (s)he only knows the context where the pattern is used. The role of AKB is meant to be descriptive in its capacity to support the architecting process. There is a strong tendency in architectural knowledge management that SHARK tools are not meant to replace people (e.g. [1]). Since architecting is essentially an art-form, tools should rather encourage than limit architect's creativity [22].

3.2.2. Mapping quality properties to architecture

As explained earlier, the quality property specification defines quality properties unambiguously for a software system. In order to benefit from these quality properties they have to be mapped to the elements of the software architecture. In other words, quality properties must be connected to the elements of the UML model.

Quality properties are stored to the UML profile as stereotypes. Thus, a UML modelling tool is able to handle quality properties like any other stereotypes. However, making quality properties mapping possible to each model element is not reasonable at the moment, and thus we restricted mappings for component, class, and object elements. Components are structured hierarchically, and this structure can be utilised in the quality mappings for defining the scope of a quality property, i.e. a quality property in the higher level reflects all lower levels. Although mappings only to components, classes and objects are supported so far, the applicability of stereotypes to associations and interfaces can easily be extended, if necessary.

In practice, the architect selects a QA profile, e.g. a profile including security properties, and loads it to the UML model. After that, the architect is able to map the quality properties to component, object and class elements. For instance, the security properties mentioned earlier can be mapped as follows: *use authentication* to the Login component and *use access control* to the Data storage component. Thus, QA profiles are used for mapping quality properties into architectural elements with UML tool. Effort required for the quality mappings depends on how specific quality properties are defined and into which level these will be mapped in the UML models. Mapping very specific quality properties for low level components (deep in a component hierarchy) is a challenging task. However, the experienced software architect is able to perform this task in the same time when constructing the architecture with UML tool.

3.3. Model based quality evaluation

The quality aware architecting approach is an iterative and incremental approach containing three main steps for quality evaluation (the bottom part in Fig. 1): (1) The quality attributes prioritized with high importance are first evaluated separately, and thereafter, a tradeoffs analysis is made between them. (2) The quality attributes with medium importance are then evaluated in a similar way after the high priority quality requirements are met. (3) Finally, the qualities, which are ranked to have only low priority, are evaluated and considered as 'nice to have' qualities.

Quality evaluation may focus on evolution qualities or execution qualities or both. Evolution qualities are evaluated with scenario-based methods, which are mostly qualitative but may also provide quantitative measurements. Execution qualities require quantitative prediction and measurement methods. Also simulation methods can be used for evaluating execution qualities but these methods are not in the scope of this paper.

The bottom part in Fig. 1 presents an overview of the whole quality evaluation process. The scenario-based methods for evolution qualities are heavily based on an evaluator's skills, architectural knowledge base (AKB), i.e. styles, patterns, and earlier defined scenarios. Supporting tools make the evaluation process more effective and trustworthy. However, tools are not necessarily needed for the evaluation of evolution qualities if appropriately skilled evaluators are available. The situation is totally different with the execution qualities, where tooling is a prerequisite for performing evaluations and getting reliable results in time. In spite of what is evaluated or which method is used, the two last activities are common; (1) comparing the evaluation results with the evaluation criteria which were defined based on the quality requirements set in the requirements specification phase and (2) making proposals on improvements required in architectural models and code.

3.3.1. Qualitative quality evaluation

Evolution qualities cannot be measured at run-time. The evaluation of these qualities is based on qualitative, scenario- and model-based analysis methods, i.e. evaluation scenarios are defined when quality requirements are defined, the scenarios are taken into account when architecting and evaluation is based on architectural models. The IEE (Integrability and Extensibility Evaluation) method [37] is a good example of this approach. The mains steps in this kind of evaluation are as follows [37,59]:

3.3.1.1. The scenario categories are created. Firstly, one considers how a system may evolve in the future and recognises prospective needs with regard to changing, maintaining and reusing the software architecture. Based on this assessment, the categories of evaluation scenarios are identified. "Replacing an existing third party component" and "porting to another operating system" could be mentioned as examples of scenario categories.

3.3.1.2. The evaluation scenarios are defined. Typically, one or two scenarios are selected to represent each category. For example, a scenario where the MySQL database is replaced with a new one could represent the category "replacing an existing third party component". Scenarios may also be weighted by estimating the probability of occurrence during a certain time period.

3.3.1.3. The defined scenarios are presented in architectural models. The scenario modelling includes the following tasks: (1) selecting an appropriate views and patterns for describing the architectural changes, (2) defining matching conditions for interface evaluation, and (3) defining assumptions, architectural constraints and a design rationale for each view. The result is a description of an architecture at the point where all figurative change scenarios have been realised. The description includes a complete set of views and selected styles and patterns.

3.3.1.4. The effects of the scenarios on architectural elements are evaluated and reported. The report compares the quality characteristics of a system to the quality requirements specification. Proposed improvements and identified unsolved problems (if any) are described and are returned to the architects. This evaluation should be carried out by someone external to the architecting team, e.g. an analyst or outside consultant.

The AKB plays an important role in both the scenario modelling phase and in the evaluation phase. With regard to scenario modelling, the AKB helps the architect to select patterns for modelling the architectural changes, which are required by a scenario. The scenario models should be created at an early phase of architecture

development. The scenario models can be stored in the AKB and later reused in other architecture development projects and/or retrieved for analysis purposes. In quality evaluation, the analyst detects the reference models being used in the architecture (an existing system architecture and/or scenario models) and then searches the AKB to see which quality attributes are associated with these reference models [54]. As mentioned before, the AKB also contains a wide range of information in using patterns, e.g. instructions on implementation or tackling with common problems. As shown in the case study later, these are a good resource when analyst needs to provide improvement suggestions for the architecting team.

3.3.2. Quantitative quality evaluation

Methods for quantitative quality evaluations can be categorised into three groups [15]:

Quantitative measuring techniques – used as part of scenario-based evaluation methods in order to estimate, for example, risks, adaptability, and flexibility. The Adaptability Evaluation Method (AEM) [75] is an example of this kind of method; it assists in the selection of the optimal architecture that meets the defined adaptability criteria. Evaluation is based on adaptability scenarios, impact analysis, and complexity analysis. The adaptability degree of the software architecture is calculated based on the results of the impact and complexity analysis. Equations are explained in detail in [76].

Prediction methods – by which the quality attributes of the architectural elements and the whole software system are predicted based on analytical models. The Reliability and Availability Prediction (RAP) [39,41] is an example of this kind of method. The RAP method predicts reliability at the (1) component, (2) architecture, and (3) system levels. A component-level prediction is based on the state-based method that predicts a component's PoF (Probability of Failure) based on the component's state diagram transformed to the Markov chain model, i.e. state diagram containing probabilities for state transitions and PoF values for each state. From these diagrams component-level prediction calculates predicted PoF for the component (also called component's independent PoF value). The architecture-level reliability prediction is a path-based approach that requires a behaviour model (i.e. sequence diagrams) of the architecture and the usage profiles of the system. The method calculates the predicted PoF values for each execution path and the predicted PoF value for the whole system. The calculations are based on the components' PoF values and probabilities of transitions in the execution paths. By analysing the reliabilities of execution paths and the components reliabilities involved in a particular path, the reliability sensitivity points of the architecture can be identified. The execution order of the paths has no impact on the system level reliability.

Measurement based methods – use model-driven testing techniques for measuring quality attributes from running systems. These methods help in predicting the quality of a system, identifying the critical components of a software system, and estimating the consequences and risks of system failures. The method used in the ComponentBee [67] is an example of this kind of approach; it supports unit-level reliability testing that calculates measured PoF_M values for a component in different execution paths. It is based on: (1) a test model and (2) a test bed. The test model describes the expected behaviour for components under testing and plug-ins that collect, process, or evaluate the raw and trace data of the dynamic behaviour of the components under test. The test bed sends input messages, being defined in the use cases, for an instrumented software component and handles the output messages that the component delivers. The instrumented component records raw and trace data regarding its dynamic behaviour. The PoF_M values are finally calculated from the processed raw and trace data. The ComponentBee uses the test model and produces PoF_M values for the software components and tested use cases based upon the following steps: (1) it processes the recorded raw and trace data and composes an overall tree presentation for the concurrent execution paths of the tested components, (2) it recognises the use cases from the overall presentation, (3) it calls the evaluator plug-ins that are defined in the test model to identify those components that participate in or cause failures in the recognised use cases, and (4) finally it produces results for the software components that have participated in the recognised use cases.

3.4. Integrated development environment

The existing tool support for the whole process is presented in Fig. 4 rounded rectangle means tool and white rectangle represents exchanged data [19]. In each phase of the process a set of tools are used; Protégé and Quality Profile Editor (QPE) for modelling quality requirements, Stylebase for Eclipse and TOPCASED UML tool for representing quality properties in architectural models; and the Stylebase for Eclipse, TOCASED UML, Reliability and Availability Prediction (RAP) and CombonentBee tools that are used for quality evaluation from models and code. Most of these tools work under the Eclipse[2] which is an open, extensible and well-standardised software development environment providing an extensible application framework upon which software can be build [70]. This section will present these tools and how they interact and support the whole process.

3.4.1. Quality specification tools

Suitable tool support is required for a quality ontology definition (see Section 3.1.1). Our first goal was to find an ontology tool for the Eclipse. Unfortunately, an appropriate ontology tool was not available for the Eclipse at the time we were developing the tool chain. Thus, we had to select a standalone tool for ontology modelling. We decided to use an open source ontology editor called Protégé[3] for QA ontology definition.

The Protégé is capable of storing ontologies in the OWL format, and thus, it is possible to replace it with a new ontology editor in the future. In addition, it is important to note that new versions of the Protégé tool have been published since we started the tool chain development. For example, a demo version of the Protégé that is integrated in the Eclipse environment has already been published.[4] However, currently the standalone version of the Protégé ontology tool is used for QA ontology definition, because it is the most mature open source ontology modelling tool.

A large software system may contain a huge amount of requirements related to different QAs (see Section 3.1.2). Each QA requirement has to be transformed into one or more quality properties that the system has to have. Typically this transformation is made by hand, although tool support is needed in order to structure and manage these quality properties in an appropriate way. Based on our knowledge, there exists no such kind of tool thus far. Therefore, we implemented a tool called QPE (Quality Profile Editor) onto the Eclipse platform. With this tool, the architect can define and manage quality properties. QPE reads an OWL formatted QA ontology as an input and stores the defined quality properties to a UML profile form as an output. The QA ontology is read into QPE by utilising the Jena framework.[5] Jena is an open source Java framework that offers a programmatic environment, e.g. for manipulating OWL files. Thus, Jena makes it possible to read the OWL formatted QA ontology stored by Protégé.

[2] http://www.eclipse.org.
[3] http://www.protege.com.
[4] http://informatics.mayo.edu/LexGrid/index.php?page = protege.
[5] http://jena.sourceforge.net/.

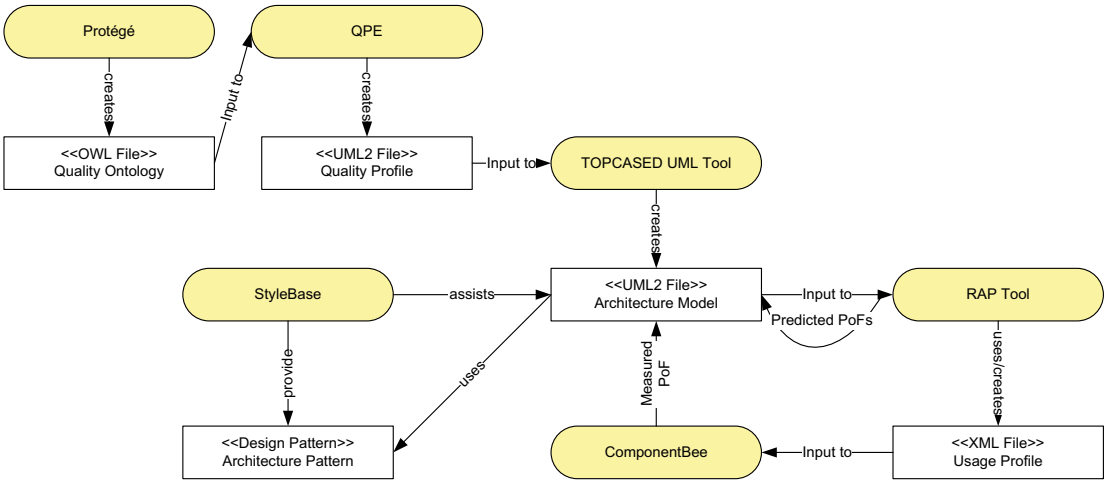


Fig. 4. An integrated tool chain for the quality aware architecting approach.

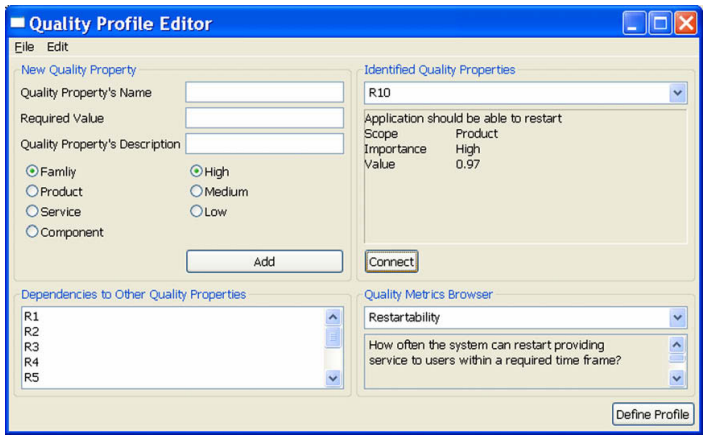


Fig. 5. Quality Profile Editor (QPE).

QPE shows (Fig. 5) the content of the metric class from QA ontology to an architect and enables him/her to inspect the meanings of the metrics. The architect enters the defined quality properties and selects a metric from the opened ontology for every quality property. Quality properties are identified by a name, e.g. *R1* and *R2*, meaning the reliability properties. Each quality property gets a textual description and a target value as a decimal number. QPE compares the entered target value and a range of the selected metric, and asks the architect to change the value if an unsuitable value is attempted. In addition, the architect can set dependencies between the defined quality properties, for instance, the reliability property *R1* is dependent of the security property *S4*. Furthermore, each quality property gets an importance level (high, medium or low) and a scope (a set of products/systems, product, service, or component). These details make it possible to define variable quality properties for the set of software products. For example, a quality property might be important to some particular product but not needed in other products. However, quality variability is out of the scope of this paper – a more detailed explanation is given in our earlier paper [57]. Finally, QPE stores the defined quality properties to the UML profile as stereotypes – containing all the information entered in QPE. Any Eclipse UML2 [10] compliant tool can now utilise the stored quality profile.

3.4.2. Architecture modelling tools

The Stylebase for Eclipse[6] [35] is an open source tool created by us for browsing and maintaining the Architectural Knowledge Base (AKB) introduced in Section 3.2.1. The tool provides functions for browsing and searching the model repository as well as for adding, deleting, and updating models in the repository. The tool supports the maintenance of both a client-side local repository accessible by one architect and a server-side shared repository to be used by a number of architects in a distributed environment [35]. The models

[6] http://www.stylebase.tigris.org.

are moved between the local and shared repositories with upload and download commands. Essentially, Stylebase for Eclipse is a user-interface of the AKB, through which existing models and their known quality properties can be reused in architecture design.

The architect can search the AKB with the Stylebase, select one or more reference models, and bring these models into the TOPCASED UML tool for further editing [20]. If the architect wants to add a new reference model into the AKB, either for self-reuse or for sharing with others, (s)he can export model(s) from TOPCASED into the Stylebase for Eclipse tool [20]. This is possible, because the Stylebase for Eclipse stores the model in an XML format that the TOPCASED tool can read [34].

After selecting one or more reference models from AKB the architect annotates the architectural models with the appropriate quality properties by using the defined quality properties stored as stereotypes in the UML profiles. The mapping of QA properties is made with the TOPCASED tool. This ensures that QA properties are available for each architect during the design time and for each software developer when implementing the software based on architectural models.

3.4.3. Quality evaluation tools

Fig. 1 showed that quality evaluation contains two separated processes, evaluation of evolution qualities and evaluation of execution qualities. From the tool support point of view, Stylebase for Eclipse concentrates on evolution qualities whereas the Reliability and Availability Prediction (RAP) and ComponentBee tools concentrate on execution qualities.

The Stylebase for Eclipse can also assist in scenario and model-based qualitative quality evaluation [34]. The analyst detects which patterns and styles are used in the architecture and uses the Stylebase for Eclipse to search the AKB for information on them, e.g. their quality properties or alternative implementation strategies (see Section 3.3.1). Currently, there is no automatic pattern detection from TOPCASED models, so this part of the task is heavily dependent on the know-how of the analyst. In addition, the Stylebase is typically used to retrieve and store scenario models from/into the AKB. *The RAP tool* [42] supports RAP method (described in Section 3.3.2). The RAP tool reads an architectural model that contains components' state diagrams, which are transformed to the Markov chain models. This transformation is made semi-automatically; the QA analyser adds the estimated probabilities of state transitions and PoF values of each state; the RAP tool adds a separate failure state and calculates new probabilities for state transitions and finally the independent PoF value for component. After component-level prediction, the RAP tool performs architecture-level prediction by utilising components' independent PoF values and usage profiles. The RAP tool offers an editor for creating and saving these usage profiles. Finally, the RAP tool calculates a PoF value for the whole system as mentioned in Section 3.3.2. All of these predicted PoF values are stored back to the architectural model as Fig. 4 shows. The main benefits for the architect when utilising these evaluation tools are: (1) the first evaluation results are available before implementation phase, (2) the results are stored in the architectural models (acting as annotations) and (3) the evaluation can be made from the existing architectural models.

In order to improve the accuracy of reliability prediction results that the RAP tool produces, it should be possible to test the reliability of used existing components and use these measured reliability (PoF_M) values in an architecture-level reliability evaluation. Therefore, we use ComponentBee testing tool [20] for measuring components' PoF_M values. ComponentBee utilises architecture-level reliability prediction models, i.e. a usage profile created by the RAP tool, in unit-level reliability testing. Finally, measured PoF_M values are stored to the architectural models as Fig. 4 shows. As an iterative manner, the RAP tool re-evaluates the architecture model by utilising measured PoF_M values. The RAP and ComponentBee tools are able to exchange information by means of a UML profile, intended for evaluation results, containing separate fields for the estimated, predicted, and measured PoF values. These fields are attached to the components of the architecture model.

4. Applying the approach

This section presents how the quality aware architecting approach was used in the development of a SMEPP middleware. We address our approach by focusing on modelling and evaluating quality properties as follows: (1) quality properties are defined for the SMEPP architecture, (2) a quality aware architecture is then modelled, and finally (3) a quality evaluation is performed at architecture and implementation levels focusing on reliability evaluation.

The SMEPP[7] project is a running FP6 project that aims to develop a secure middleware, called the SMEPP middleware, for embedded peer-to-peer systems. In peer-to-peer systems, all the elements of the network are symmetrical, and in most cases, the mechanisms of communication are based on dynamic ad hoc networks among peers. Advanced technologies of short distance wireless communications used in peer-to-peer systems have opened up new areas of applications with technological challenges. Moreover, peer-to-peer systems are extremely vulnerable to any type of internal or external attack. With these concerns in mind, the SMEPP middleware has to be secure, generic, and highly customizable, allowing for its adaptation to different devices (from PDAs and new generation mobile phones to embedded sensor actuator systems) and domains (from critical systems to consumer entertainment or communication). The generic middleware has to implement (1) new security primitives for the specific attacks in this type of system, (2) a new interaction model specific to peer systems, and (3) the necessary services to support this interaction model. Moreover, a customizable component framework and the supporting tools are developed for adapting the middleware to different embedded devices and networks.

4.1. Modelling quality requirements

First, the initial requirements for the generic middleware were specified and divided into functional and non-functional categories. In total, 43 functional and 83 non-functional requirements were identified. Each requirement was associated with a unique identifier and mapped into the components and conceptual layers of the middleware architecture. The importance of a requirement for each architectural layer was defined and expressed using three categories: high, medium and low. The related quality attributes were also identified. Table 1 exemplifies the description of the non-functional requirements. The entire requirement specification can be viewed online[8].

The initial quality requirements were enhanced as explained in Section 3.1. We derived quality properties from the initial quality requirements and selected appropriate metrics for each quality property. Table 2 shows an example of the derived quality properties and metrics related to reliability. As mentioned in Section 3.1.2, we have two quality ontologies ready at the moment. The reliability ontology is used as an example in this case study.

The software architect uses the Quality Profile Editor (QPE) (Fig. 5) and defines the above-mentioned quality properties to the quality profile form in the following steps. The architect:

[7] http://www.smepp.org/.
[8] http://www.smepp.org/Deliverables.aspx.

Table 1
An example of the initial quality requirements of SMEPP.

Req. ID	Category	Description (from D1.1, D1.3)	Related conceptual layer(s) of MW architecture	Importance	Related quality attribute
HS.APP.MID.11	Middleware non-functional application	Middleware must be stable enough in all possible situations (reconfigurability of the network, disconnection and reconnection of nodes, different network interfaces, etc.)	Applications and services X Extensions and service model support X SMEPP common services X SMEPP enabling services X Infrastructure	 High High High High	Reliability adaptability

Table 2
A set of quality properties and metrics derived from the initial quality requirements.

Requirement ID + description	Property ID + description	Related metric
REQ.MID.REL.1: The middleware must endure the loss of communication of some nodes and store the unprocessed information in the network	R1: The middleware must be able to restore itself in 90% of cases	Restorability
REQ.MID.PER.1: The middleware must provide a timely response of less than 10 s for an alarm generated by a peer (a sensor)	R2: Mean recovery time should be less than 5 s R3: Availability should be at least 98%	Mean recovery time Availability
HS.APP.QoS.14: QoS in the video streaming	R4: Availability of the video streaming component should be at least 92%	Availability
HS.APP.MID.11: Middleware must be stable enough in all possible situations (reconfigurability of the network, disconnection and reconnection of nodes, different network interfaces, etc.)	R5: Mean time between failures of the middleware should stay above 12 h. R6: Middleware should contain mechanisms for breakdown avoidance for 85% of recognised failure cases	Mean time between failure Breakdown avoidance
EM.APP.PER.6: The application must provide data and services in real-time	R7: Availability of the application should be at least 95% R8: The application should contain mechanisms for breakdown avoidance for 80% of recognised failure cases R9: Mean down time of the application should be less than 10 s R10: The application should be able to restart in 97% of breakdowns cases	Availability Breakdown avoidance Mean down time Restartability

4.1.1. Selects a quality profile folder

The quality properties from profiles in this folder are listed in 'Dependencies to Other Quality Properties' group, in order to set dependencies to the earlier defined properties.

4.1.2. Opens quality ontology

The metrics are shown in the 'Quality Metrics Browser' group.

Enters the defined quality properties – by utilising 'New Quality Property' group. The entered quality properties appear to 'Identified Quality Properties' and 'Dependencies to Other Quality Properties' groups.

Connects each of defined quality properties to an appropriate metric (from metrics group) and any possible other quality properties (from dependencies group) – by using the 'Connect' button.

Fig. 5 represents a phase when the architect is selecting a quality metric (Restartability) for the quality property R10 (see Table 2). Lastly, QPE creates a UML profile and stores the defined quality properties into it in a stereotype form. When the quality profile contained all quality properties related to reliability, another quality profile was created for security properties. Thus, quality profiles are quality attribute specific. In the architecture modelling phase, the reliability and security properties are then mapped to UML models.

SMEPP had 83 initial quality requirements that resulted in over 100 quality properties. The reason is that several quality properties are derived from one initial quality requirement as seen in Table 2, where the EM.APP.PER.6 requirement ends up four reliability properties, measured by different metrics. Thus, by QPE the architect can refine and manage quality requirements in a systematic way and provide the results as reusable design artefacts. Moreover, the architect can later on add new quality properties to the defined profile or create an entirely new quality profile, if an added requirement is related to a quality attribute not covered by the existing profiles.

4.2. Representing quality in architectural models

The TOPCASED tool and UML2 notation were used for modelling the SMEPP middleware architecture. The architecture was modelled from four different view points (structural, behavioural, deployment, and development views) and on two levels of abstraction (conceptual and concrete) in accordance with the QADA®[9] (Quality-driven Architecture Design and quality Analysis) methodology. The purpose of this chapter is not to present the architectural design of the SMEPP platform (as already done in [46] and [47]), nor to explain the details of QADA methodology (as already done, e.g. in [50]). Instead, we exemplify how quality requirements were transformed into architectural models as described in Section 3.2. Firstly, the evolution quality requirements were taken into account when selecting styles and patterns to be used in the architecture. Secondly, the execution qualities were mapped into architectural elements.

In the model selection phase, the Stylebase for Eclipse tool should be used to search for different design alternatives from the AKB. As explained earlier, the AKB represents the mappings between the evolution qualities and design decisions. The architect searches the AKB by the desired quality attributes and selects patterns and styles on that basis. Thus, the architect still needs design knowledge in order to make the decision which style or pattern to select. The AKB helps in making decisions by providing knowledge on earlier design decisions represented as models but it does not have intelligence to make design decisions on architect's behalf. If it is discovered in the quality evaluation phase that the selection does not meet the quality requirements, another option is selected when the architecture is redesigned. Redesign can be made manu-

[9] QADA is a registered trademark of VTT, http://virtual.vtt.fi/virtual/proj1/projects/qada/.

ally or semi-automatically by exploiting a model transformation technique [54].

The architecture of the SMEPP middleware was designed by our partners, who did not use the Stylebase for Eclipse tool (or any other knowledge base solution) at the time. Design decisions were based on their own knowledge and experience instead. The primary architectural styles and patterns were selected as follows [46,75]: (1) the Layers Pattern [11] and (2) the Microkernel Pattern [11] to depict the layers of the architectures, (3) General Communication Middleware (GCM) [66] to solve the adaptive middleware issues, (4) a component-based architecture paradigm [64] to depict the architectures, and (5) connectors [12] to depict the relationships between the architecture components. Later on, the Stylebase for Eclipse was used in the quality evaluation of these decisions.

Execution qualities were made visible in architectural models. Once initial design decisions had been made, the structure of the SMEPP middleware was outlined in an UML2 Component Diagram. The quality profile, which was earlier created with the QPE tool (see Section 4.1), was opened in the TOPCASED tool. The quality properties were then mapped into components of the architectural model as stereotype values of the components, classes and objects.

Fig. 6 presents how an architect browses the imported quality profile and maps quality properties (saved as UML stereotypes) into architectural elements. The architect maps these quality properties to architectural elements by utilising the default functional-

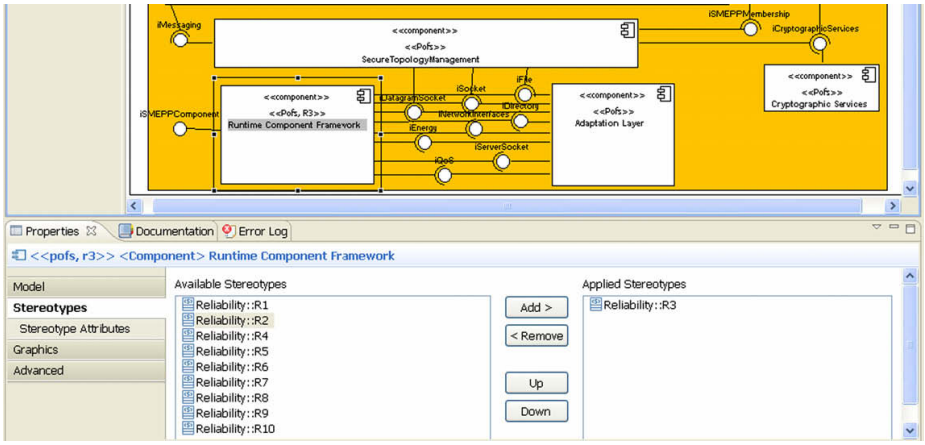


Fig. 6. Mapping quality properties to the components.

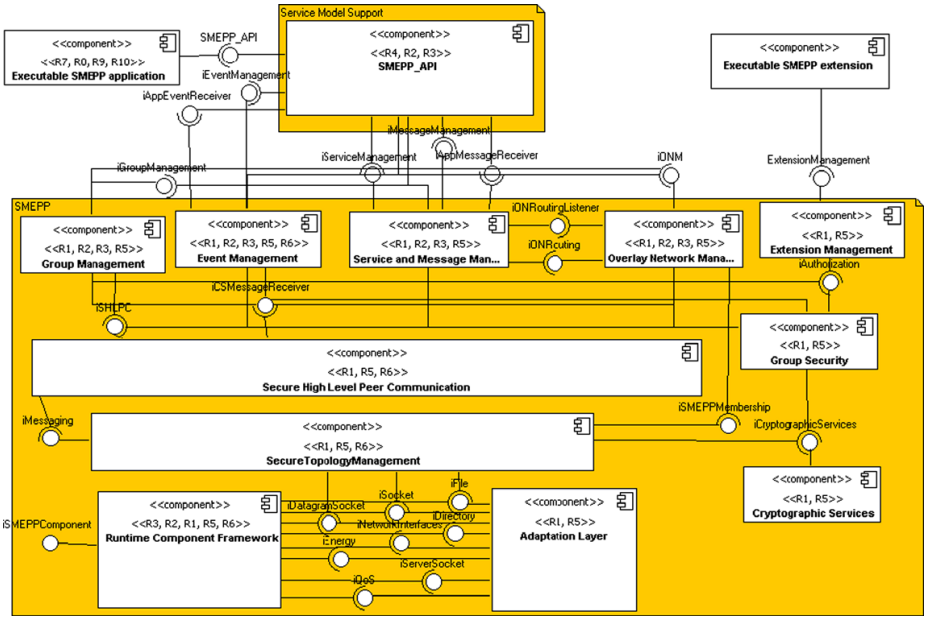


Fig. 7. The structural view annotated with quality properties.

ity of the UML tool. Hence, the mapping depends on the experience of the architect, as mentioned in Section 3.2.2. Some quality properties from Table 2 relate directly to one particular component, whereas some are more generic ones and related to several components. For instance, quality properties R7, R8, R9, and R10 are mapped only for *Executable SMEPP application* component. On the contrary, the quality property R3 that is related to availability is mapped to six components, meaning that availability of the whole middleware requires that availability of all these components is in the sufficient level. Similarly, other quality properties are mapped to the components which are responsible for achieving these quality properties. The resulting architectural description, which contains measurable quality properties as stereotype values of components, is presented in Fig. 7.

4.3. Model based quality evaluation

The interest of the QAs for a software architecture is in how they interact with, and constrain, each other, and how they affect the achievement of other QAs. Therefore, a set of QAs must be handled at the same time and tradeoffs between them calculated and managed. However, the developers must first ensure that the used architecture, selected components, and the software system itself meet the desired quality requirements.

In the following sections, we concentrate on three kinds of evaluations; (1) a scenario-based evaluation of evolution qualities, (2) a prediction-based evaluation of execution qualities, and (3) measurement-based testing. The following sections present how these evaluations were carried out and how the tool suite worked in the case in hand.

4.3.1. Qualitative evaluation of evolution qualities

Qualitative, scenario-based evaluation methods are used to assess evolution qualities as explained in Section 3.3. The integrability, extensibility and portability of the SMEPP middleware architecture were evaluated by using such a method, namely IEE (Integrability and Extensibility Evaluation) method [59]. The purpose of this chapter is not to present evaluation results, nor to describe the IEE method as already done in [37]. Instead, we briefly exemplify the evaluation processes and highlight the benefits of using the Architectural Knowledge Base (AKB) and the related knowledge management tool, Stylebase for Eclipse.

Firstly, one has to decide which quantitative quality requirements are selected for evaluation and derive concrete *quality goals* from selected requirements. Quality goals are prioritized so that the evaluation can start from the most important ones. In essence, they answer the question *how well* something is to be achieved. Secondly, the *scenario categories* are derived from the quality goals and at least one *change scenario* is defined per each scenario category. Quality goals and scenarios should be ideally defined at the same time when the quality requirement specification is written. In the case SMEPP, this task was performed later, when the tool chain was taken into use. Table 3 presents a portability related scenario, which is taken as a example herein. The whole process of determining scenarios and quality goals is throughout explained in [37].

Once in the example scenario, SMEPP would be ported into a Mote-class hardware platform for sensors. A sensor is a very tiny device with extremely constrained resources of memory, battery, and computing. A sensor could not face the technical problems arising in the implementation of the whole SMEPP specification [79], and therefore, a new "light" version of SMEPP would be required. SMEPP Light would take the special characteristics of Motes into account and offer only a minimum set of functionality. In the scenario modelling phase, the analyst designed the conceptual architecture of SMEPP Light and modelled it with the TOPCASED tool. In the beginning, the Stylebase for Eclipse tool was used to search for reusable scenario models in the architectural knowledge base (AKB). In this case, suitable scenario models were not found, and therefore, the conceptual architecture of SMEPP (Fig. 7) was used as a starting point. The analyst imported the model (Fig. 7) from Stylebase for Eclipse to the TOPCASED tool and modified it to create SMEPP Light. Table 4 illustrates the essential differences between SMEPP and SMEPP Light.

While modelling, the analyst encountered some problems. For example, SMEPP Light does not support services and its communication is event based. Therefore, it could not process messages from conventional SMEPP devices such as PDAs. An adaptor component would be needed to enable full communication between "normal" and "light" versions of SMEPP. At this point, the analyst used Stylebase for Eclipse to search for solutions. Firstly, he searched for a ready solution: someone could have already designed a domain-specific micro-architecture to target this issue. In this case, there was none. The analyst continued by searching information on generic patterns (Fig. 8), reading about their applications and quality consequences. After weighting different options, the analyst selected the combination of Proxy [11], Adapter [27] and Forwarder–Receiver [27] patterns to form the basis of the new adaptor component. The skeletons of these patterns were dragged-and-dropped from Stylebase for Eclipse into the TOPCASED tool and the analyst drew the micro-architecture of the new component.

At the end of the scenario modelling phase, the ready scenario models (i.e. the models of SMEPP Light and the adaptor component) were exported and saved into the shared repository of the Stylebase for Eclipse. In addition to being used in analysis, the models could be reused as a design document later on, i.e. if the scenario is realised, a ready implementation plan is at hand. In

Table 4
Components of SMEPP and SMEPP light.

High-level components	SMEPP	SMEPP light
SMEPP API	Yes	Limited
Group management	Yes	Yes
Event management	Yes	Yes
Service management	Yes	No
Overlay network	Yes	No
Peer-communication layer	Yes	Yes
Topology management	Yes	No
Adaptation layer	Yes	Yes
Security and crypto services	Yes	Limited

Table 3
Example definition of a quality goal and related scenario.

Requirement ID + description	Quality goal	Scenario category	Scenario
MID_NON_FUN3: The middleware system architecture must be implementable on a variety of heterogeneous operating systems and hardware platforms	Porting to another platform requires moderate programming effort and does not break existing architectural style(s)	Porting the SMEPP middleware to a new operating system and/or hardware platform	SMEPP is ported into a Mote-class hardware platform for sensors (and, consequently, to a TinyOS operating system)

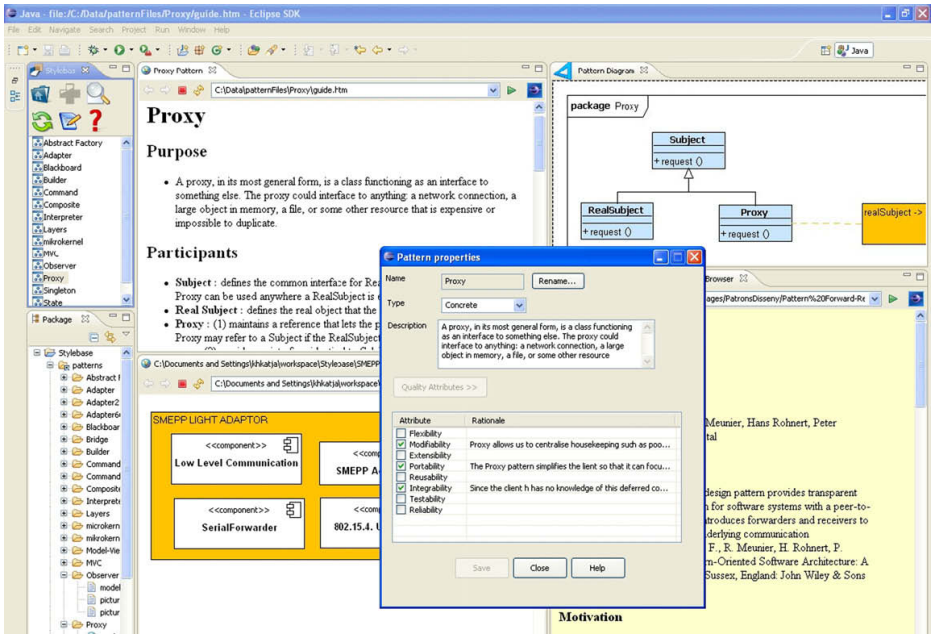


Fig. 8. The Stylebase for Eclipse is used to search AKB.

addition, the micro-architecture of the adaptor component is quite generic and can be reused in another development project.

Once the scenarios had been modelled, the analyst was ready to evaluate how the predefined quality goals are met in the architecture. Layers and Microkernel are the main architectural patterns that are used in the SMEPP middleware architecture. Firstly, Stylebase for Eclipse was used to find some general information on these patterns. For example, the AKB contains information on how the Layers pattern supports *modifiability*, *portability*, and *reusability* and, in turn, how the Microkernel pattern supports *modifiability*, *extensibility*, *portability*, and *scalability*. Fig. 8 (see the Pattern Properties windows in the centre) illustrates how Stylebase for Eclipse presents (a) the list of quality attributes supported by a pattern and (b) the description on why and how the given qualities are supported.

The idea of the scenario-based evaluation is to analyse the scenario models and assess the architectural impact of the figurative scenarios. Each pattern in the AKB is associated with usage instructions (see the HTML-formatted "guide" in the top-left corner of Fig. 8) that include a section on dealing with common problems. Therefore, when scenario evaluation relieves a specific problem, the Stylebase for Eclipse tool can be used to search for suggestions on how to improve the architecture. In the sample scenario, the main architectural styles, Layers, remained intact. However, there would be major changes to the behaviour of one layer, Common Services, and this could turn into a snow-ball effect, impacting several other layers. The Stylebase for Eclipse provided instructions on how to tackle the problem of cascades of changing behaviour in the Layers architecture. The sample scenario also demonstrates the breakage of Microkernel architecture, because the new light version of SMEPP is much smaller than the original functional core of the SMEPP Microkernel. The Stylebase for Eclipse provided information on the problematic of determining the core functionality in the Microkernel architecture. These are examples of the issues which were mentioned in the evaluation report, accompanied with improvement suggestions.

4.3.2. Predicting reliability

A software architect or analyst is able to predict reliability of software with the RAP tool. Firstly, (s)he has to construct a component diagram, sequence diagrams and optional state diagrams as a Markov chain form [13]. Naturally, mostly these diagrams are produced during the architectural modelling phase and only little additions are needed in this phase. The component diagram presents a concrete static structure, i.e. the components and their interfaces, of a software system. Instead, sequence diagrams present behaviour, i.e. execution paths for the components of a software system. The component's internal behaviour is modelled by means of state diagrams, which are converted to the Markov chain form. In the Markov chain form each state gets a probability of failure (PoF) value and each state transition gets a transition probability value. From the Markov chain model the RAP tool calculates an independent PoF value for the particular component. Constructing Markov models is not mandatory. For example, for small components or third party components the software architect can estimate a component's independent PoF. After the RAP tool has gathered a components' independent PoF values – either by calculating from the Markov models or as an architect's estimate – the components' PoF values are calculated in each execution path and in the whole system. By utilising these values, the RAP tool calculates a PoF value for each execution path, and finally for the whole system as a weighted average from execution paths' PoFs.

The reliability prediction utilises the diagrams the architect has defined with the TOPCASED UML tool. First, the analyst took the component diagram of the SMEPP architecture and connected a special UML profile to it in order to store the estimated PoF values and results from the RAP tool into this profile. In that way, the analyst was able to see components' PoF values directly from the UML

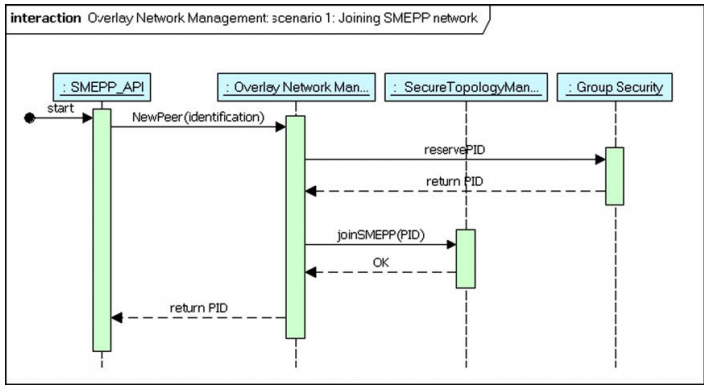


Fig. 9. Joining SMEPP network sequence diagram.

diagrams. Thereafter, separate sequence diagrams were defined for each use case. The use case "joining SMEPP network" is given as an example in Fig. 9. In the sequence diagram a lifeline corresponds to the particular component from the component diagram.

After the sequence diagrams have been completed, the architect or analyst connects a particular behaviour, i.e. state diagrams converted to the Markov chain forms, to the components. However, in this case the analyst decided to give an estimated PoF values for each component – instead of the Markov chain models enabling him/her to get the first reliability prediction very quickly. The analyst based the estimated PoFs on the architects' initial insights of the SMEPP middleware and the anticipated complexity of the components to be implemented. The architects estimated (a) an independent mean time between failure (MTBF) and (b) usage frequency for each component. From these values, the analyst calculated the PoF value for each component. For example, Event Management (EM) component's MTBF was estimated to be 24 h and its usage frequency every fifth second. This means 17,200 execution times per 24 h. Thus, the independent PoF value for the EM component is estimated to be 1/17,200 = 0.0000581 rounded to 0.00006. The Independent PoF column in Table 5 lists the estimated independent PoF values for the components of the SMEPP architecture.

Now, the UML model of the architecture contained the estimated PoF values, too. Next, the analyst defined the usage profile(s) for the system. The usage profile denotes execution times for each sequence diagram. From these execution times, the RAP tool calculates an execution probability for each sequence diagram. The RAP tool makes it possible to define many usage profiles to be analysed separately. The analyst can construct separate usage profiles for different stakeholders or for an entirely different context of the system usage. For example, in the SMEPP case there can be usage profiles for observing a nuclear power plant, monitoring patients' health information in the health care environment, and a usage profile representing a default (an average) use of the system. Fig. 10 resents an average usage profile that is created with the usage profile editor of the RAP tool.

Figs. 11 and 12 show the results the RAP tool has calculated when the usage profile from Fig. 10 was used. Fig. 11 shows the results from the components point of view, i.e. components' execution times, independent PoF values and system dependent PoF values. The system tab of the result window (Fig. 12) presents the results from the system point of view and lists execution paths and their execution times, PoF values, descriptions, and components. In addition, the probability of failure for the entire system is shown. Columns 3rd and 4th in Table 6 list the execution times

Table 5
Summary of components.

Component's name	Independent PoF	Execution times	System dependent PoF
Adaptation layer	0.0	0	0.0
Cryptographic services	0.00006	0	0.0
Event management	0.00006	41,400	0.000002
Group management	0.00012	86,700	0.00001
Group security	0.00012	2000	0.0
Overlay network management	0.00022	11,200	0.000002
Runtime component framework	0.000012	900,000	0.000011
Secure high level peer communication	0.00006	115,100	0.000007
Secure topology management	0.00022	2500	0.000001
Service and message management	0.00022	67,720	0.000015
SMEPP_API	0.00006	139,500	0.000008

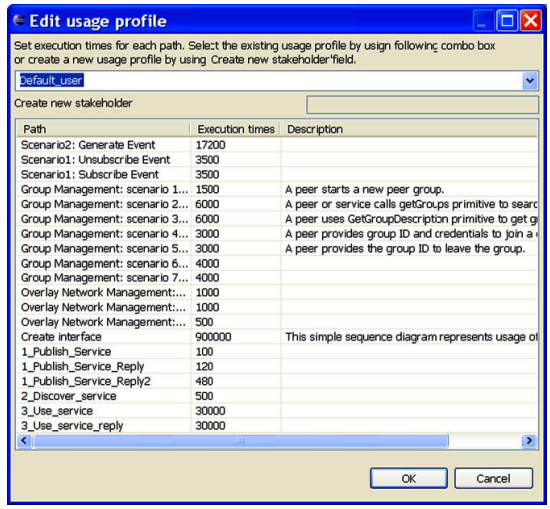


Fig. 10. The usage profile editor of the RAP tool.

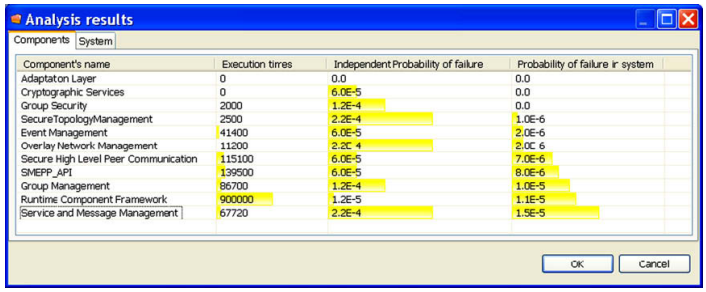


Fig. 11. The PoF values of components as analysis results.

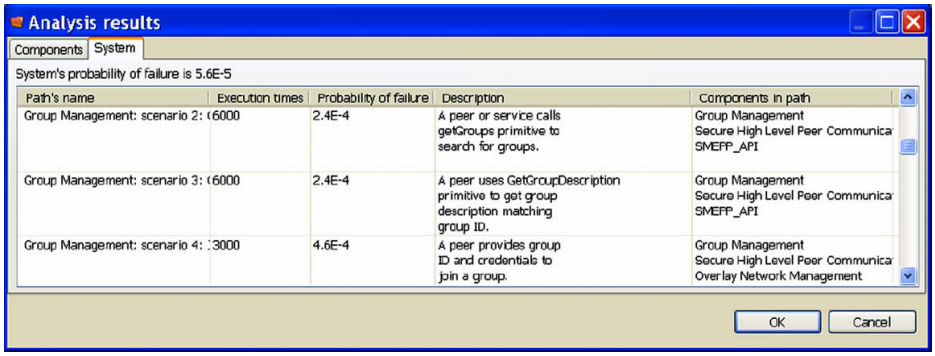


Fig. 12. The PoF values of paths and the system as analysis results.

and system dependent PoF values that are calculated for the selected usage profile (Fig. 10).

In conclusion, Table 5 lists all components, (estimated) independent PoFs, execution times, and the system dependent PoFs for the usage profile from Fig. 10. The first notable thing from these results is the PoF value of the *Runtime Component Framework* component. The architect of the SMEPP system assumed that it is the most unreliable component of the system – based on its huge amount of executions. However, the analysis revealed that the *Service and Message Management* component is even more unreliable and that the *Runtime Component Framework* comes after that, as the results show. Thus, the system's reliability can be influenced the most effectively by enhancing the reliability of the *Service and Message Management* component. The second issue, raised when the architect inspected the results, is the execution times of the *Secure High Level Peer Communication* component. The component is the third executed component of the system after *Runtime Component Framework* and *SMEPP_API* components. Based on the analysis, the probability of failure to the whole system is predicted to be 5.6E−5, and the result corresponds to the architect's assumption very well.

As mentioned earlier, the RAP tool stores predicted reliability values to the UML model by utilising the designed UML profile. Thus, these values can be also examined from the TOPCASED UML tool. Thus, the architect can see the component's independent PoF value given by the architect and the system dependent PoF that the RAP tool has calculated. In addition, the profile contains fields for the results calculated by the Markov model and the measured PoF value, measured by ComponentBee.

4.3.3. Measuring reliability

Many software projects have multiple participants that provide components for a software system. For example, reusing older software components is a common method to decrease the implementation effort of new software systems. Unfortunately, these components can cause unexpected problems. For example, our first plan was to test the Event Management (EM) component of the SMEPP on the system level. More precisely, our plan was (1) to implement a test bed that calls the SMEPP system to execute the test sequence, (2) to insert probes to observe and to record trace and raw log data of the SMEPP system, and finally (3) to evaluate the recorded raw log file and to calculate measured PoF values for the EM component in various use cases. Unfortunately, the SMEPP system did not work correctly when we tried to execute it with the profiler tool of the Eclipse. In addition, we did not have source code for all the components of the SMEPP system. Thus, it was difficult to trace the reason why the execution of the SMEPP system failed in the profiler tool. Threads may be one possible reason for the observed problems. The profiler tool may affect the scheduling of threads that perform the configuration sequence of the SMEPP system. As a result, the SMEPP system is not necessarily correctly initialized, which leads to the malfunction of the system. In order to avoid the described problems, we decided to implement a separate test bed for the EM component.

The testing was implemented according to the following steps:

1. A test model was created for the EM component with the ComponentBee. It defines a *"test sequence"* use case and the messages that are delivered in the use case.

Table 6
Summary of the applications of our approach.

Trial	Scope	Results	Type of product	Lessons learned
1	Modelling by QADA®	Middleware service architecture	Prototypical	How to define and justify multiple viewpoints and abstraction levels for service architectures
2	Modelling and evaluating evolution qualities	Multimedia streaming service, instant messaging and presence service	Industrial	How to select styles and patterns for managing modifiability, portability and extensibility
3	Modelling and evaluating variations	A family of mobile terminal software systems	Industrial	How to model quality variability inside and among products How to identify variability among a family of products How to select proper binding times for variations How to evaluate evolution qualities
4	Architecture knowledge management	Creation of reference architecture, architecture knowledge management support and domain-specific patterns for wireless services	Prototypical	How to create reference architecture How to define and share AKB in a distributed multi-national development team
5	Reliability and availability prediction (RAP)	A distribution management platform for services	Prototypical	How to annotate service architectures with variable reliability properties How to evaluate reliability of a family of service architectures
6	Evaluation of a product family	The approach applied to a product family of mobile network analysers Reusable domain-specific patterns defined	Industrial	How to elicit domain-specific patterns from existing documents and source code How to evaluate evolution qualities and their variations from architectural models
7	Modelling a family of measuring systems	Service architecture with high quality requirements; reliability, security, maintainability, portability, flexibility	Industrial	How to elicit and manage a large set of quality requirements and their variations in a software product line How to transform quality requirements to architectural models
8	Adaptability evaluation	Service architecture of wireless environment controlling systems	Industrial	How to evaluate adaptability of software architectures How to apply the AEM method to an industrial case study
9	Integrability and extensibility evaluation	IEE method applied to an open source software development	Open source	How to apply the IEE method for refining and extending existing open source software components
10	Reusable models of quality attributes Quality variability model	Service architecture of a family of weather measuring systems	Prototypical	How to define security and reliability ontology How to define a common quality variability model for execution qualities How to create modelling techniques and tools for defining quality attribute ontologies How to use quality attribute ontologies for defining metrics for quality requirements How to map quality properties to architectural models How to combine reliability prediction models with model based dynamic testing
11	Validation of reliability evaluation and the integrated tool environment (RAP and ComponentBee)	Personal information repository, a business-to-consumer health-care document delivery system	Industrial	How to apply the quality driven modelling methodology and supporting tools for industrial cases How to define stakeholders based usage scenarios for evaluating reliability from architectural models How to transform predicted and measured quality properties between models and source code

2. We created a Java project in the Eclipse workspace and coded a test bed for the EM component with Java. The test bed starts three threads. Each of the threads executes a test sequence that calls the send message, push message, subscribe event, and quit event subscription methods of the EM component. Each thread repeats the test sequence 1000 times.
3. We used the test model editor of the ComponentBee and adapted the test model for the implementation components by adding adaptor elements to the test model. The adaptor elements define which methods are called when a particular test model's message is passed. Secondly, we implemented evaluator plug-ins that are capable of recognising the failed messages and defining which software components caused failures in the executed use cases.
4. A TPTP ProbeKit[10] was generated for the test model with the ComponentBee. We started the profiler tool of the Eclipse. The tool inserts the probes of the ProbeKit to the components under testing and runs the instrumented Java program that finally produces the raw log file.

[10] http://www.eclipse.org/tptp/.

5. The raw log file was pre-processed with the ComponentBee. The ComponentBee took the raw log file as an input and composed an overall presentation (a behaviour tree) for the test sequences that were executed in different threads. This tree contained only messages that were defined in the test model.
6. Behaviour patterns were extracted from the overall behaviour tree according to the following steps: (1) The ComponentBee generates a BNF grammar [28] that defines production rules for the behaviours that are described in the test model, then (2) calls the Java Compiler-Compiler (JavaCC)[11] to generate a *BehaviourParser* for the grammar, and finally (3) calls the generated *BehaviourParser* to compose a symbol tree (a behaviour pattern tree) of the behaviour tree.
7. The behaviour pattern tree was finally evaluated with the ComponentBee that activates the defined evaluator plug-ins to evaluate and write the evaluation results to the nodes of the behaviour pattern tree. The behaviour pattern tree is written to an XML file that contains both the evaluation results and the extracted behaviour patterns.

The parser extracted 2999 test sequences from the behaviour tree. In addition, one unidentified behaviour sequence was extracted from the behaviour tree. We evaluated the unidentified behaviour sequence and noticed that a synchronisation problem exists in the EM component. More precisely, an *ArrayIndexOutOfBoundsException* is thrown if a thread is trying to call its push method while another thread is calling its unsubscribe event method. Thus, the measured PoF value for the EM component in the "test sequence" use case was 0.0003.

In order to facilitate the iterative development of the software system, the measured PoF values are imported to the UML models by utilising the UML profile. The architect/analyst re-evaluated the architecture with the RAP tool. This time, instead of estimated PoF values, the more accurate PoF values measured with the ComponentBee are used in architecture-level reliability prediction. The re-evaluation produces a new predicted PoF value for the whole architecture of the SMEPP system and new predicted system dependent PoF value for EM component.

Our experiences in the reliability testing of the SMEPP system showed that many unexpected issues may affect or even prevent the testing of the target software components. For example, if threads are used in a software system, the usage of a profiler tool may affect the scheduling of threads and even prevent the usage of the software system. However, in a best case scenario, these kinds of problems can reveal bugs related to thread synchronisation in a software system.

It took four working days to implement the reliability test and evaluation for the event management component of the SMEPP system. The test model creation took a few hours. The ComponentBee evaluated the recorded raw log (the size was 61 Mb) and produced reliability values for the components in 9 min and 30 s. Of course, the raw log size has a great impact on this time. The ComponentBee prototype records the raw log data in an uncompressed XML format, which leads to quite large raw log files. Large scale testing requires very fine-grained raw log writers that only record the required raw and trace data to the raw log. Additionally, a more efficient format is needed for the raw logs.

4.3.4. Re-predicting reliability

After measuring the reliability of the Event Management component, as described in the previous section, the RAP tool utilises this measured PoF value in re-predicting reliability of the SMEPP system. Thus, the re-prediction produces more accurate results because one measured value (produced by the ComponentBee) is used instead of initial estimates (given by the architects of the SMEPP).

Re-predicting reliability from the architectural model does not differ from the procedure represented in Section 4.3.2. However, at this time the analyst constructed a new sequence diagram that conforms to the used test model because this behaviour was detected during the testing, and the prime model did not contain this sequence. Thereafter, the analysis selected by the RAP tool that the measured PoF value was used instead of the estimate. Lastly, the RAP tool gave the refined results in the same format as earlier (cf. Fig. 13).

As we can see from Fig. 13, the results of the re-prediction are close to the results of the first prediction. In the first time, the PoF value in SMEPP system for Event Management component was 0.000002 and re-prediction produces a PoF value of 0.000003. PoF values for other components are not changed because original initial values were used for them. In addition, the PoF value for the whole SMEPP system did not change in this iteration, and re-prediction produced the same PoF value for the whole system as was the case the first time (5.6E−5). The reason why the PoF value for the whole system was not changed this time can be seen easily. The Event Management component's new PoF value differs only slightly from the first value and because there are more dominant components in the system, this difference is not visible in the PoF value of the whole system. In other words, the result is the same but we can be more confident about its correctness.

In conclusion, re-prediction of reliability can be made very quickly. An architect/analyst is obliged to add only one additional sequence diagram to the initial model, and thereafter, the RAP tool performs the prediction automatically. However, a new diagram is not needed in ideal cases as the initial architectural model is comprehensive, i.e. implementation corresponds to the architecture design. In the next iteration, a new component is implemented, tested, and again the evaluation results came back to the RAP tool for re-predicting reliability of the whole SMEPP system.

5. Discussions

As a benefit of our quality aware software architecting approach is the fact that it facilitates software developers to: (1) model reusable quality requirements for software systems, (2) transform the modelled quality requirements to the architectural models, (3) reuse existing design knowledge such as architectural styles, generic design patterns, and domain-specific patterns to achieve desired quality goals in software architecture, and finally (4) evaluate that the desired quality goals are met in the software models and code. This all helps software developers to manage both the quality knowledge and the architectural knowledge during the construction of a software system. In addition, quality evaluation task is supported by providing method support for both qualitative and quantitative quality evaluation. For qualitative quality evaluation there is the Integrability and Extensibility Evaluation (IEE) method whereas for the quantitative quality evaluation there are the Adaptability Evaluation Method (AEM), the Reliability and Availability Method (the RAP method), and the measurement method used in the ComponenBee.

5.1. Maturity of the approach

Our approach was developed incrementally and applied to several software developments in the international joint research projects and national contract research projects. Our approach is intended for the development of complex long-lasting software intensive systems, e.g. networked embedded systems. Table 6

[11] http://www.javacc.dev.java.net/.

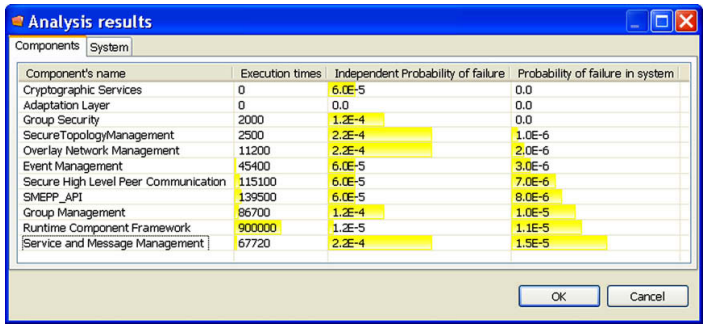


Fig. 13. The PoF values of components after the re-prediction.

shows that each trial resulted in an industrial or prototypical type of product. Industrial products were developed together with companies and aimed at use in commercial product developments. Prototypes were developed with commercial products in mind proving feasibility studies, e.g. for technology investigation. The methods and techniques were initially developed in joint research projects and applied to prototypical products. Thereafter, the developed parts were applied to and validated in the industrial product developments, which have also fed identification of new problems to be solved. Table 6 also summarises the main lessons learned in each trial. For example, in Trial 3 we learnt why and how variations do arise in product families and how they should be managed in architectural models. We also identified that managing quality variations is of high importance. In Trial 7 several stakeholders were involved in requirements specification that resulted in a large amount of quality requirements. Thus, we realised in a concrete way how important it is to have appropriate tools that assist in defining and managing variable quality requirements and their relations. The development of AKB (Trial 4) origins from the needs of the distributed development team of an international joint research project where our target was to define a generic reference architecture for wireless services. Furthermore, after the evaluation methods were demonstrated by laboratory cases they were adapted to industrial settings and validated in industrial or open source product developments (Trials 2, 3, 6, 8, 9, and 11). We selected an approach to integrate ontology orientation with model driven architecting. The selection seems to be the right solution; nowadays industrial companies are more and more interested in applying ontologies for capturing domain knowledge to reusable models. In our case, a quality attribute formed a domain. Knowledge of any domain can be defined as an ontology or reusable patterns. We have identified and defined reusable domain-specific patterns for wireless services and mobile network analysers (Trials 4 and 6). However, our recent work concentrates on applying ontology orientation for defining domain knowledge for achieving interoperability between cross-domain architectures of different application areas, e.g. applications of personal spaces and a smart city.

One of the most important factors on maturity of the approach is the availability of supporting tools. Meanwhile our approach matured, several tool developments were carried out.

Table 7 summarises the maturity levels of our achievements in the tool development. In the very beginning of their life cycle, all tools were tested in laboratory cases. Thereafter, all tools except QPE were individually applied into real-life software development projects. ComponentBee and Stylebase for Eclipse were also advertised as open source projects and we received dozens of improvement requests. The tools were incrementally improved on the basis of gained experiences and feedback. In total, there were 12 different tool versions: one QPE version, five Stylebase versions, three RAP versions, and three ComponentBee versions. Each version enhanced the previous versions, thus offering better support for the whole approach.

Gained information from different tool implementations:

- From the QPE there is the initial version available, although we know that the tool requires many usability related enhancements. However, the first version has also indicated that metric selection has to be facilitated in the future.
- The first version of Stylebase for Eclipse was dependent on a closed source modelling environment, which impaired extensibility and limited potential user base. The open source version was designed from the view point of extensibility and integrability, which are still the strongest advantages of the tool [36]. The later releases implemented several usability requests, which we received from users. However, many challenges remain. Knowledge management has to be made easier, for example.
- From the RAP tool the first version revealed that a separated simulation model is difficult to use and maintain. Thus, the second version utilised only UML2 compatible models, which makes the prediction much easier for the analyser. Leaving the simulation model out affected also the RAP method and the preceding architecture modelling phase. Therefore, this is a good example how the tool development has offered valuable feedback also for the whole approach.
- Our goal was to publish the ComponentBee tool as an open source project under the Eclipse technology project. The ComponentBee project proposal has now gone through a review process and is now accepted and published. In order to follow the base idea of Eclipse that means that Eclipse projects do not just develop tools and runtimes but they also develop extensible frameworks for building, deploying and managing software across the life-cycle, we divided the ComponentBee tool into: (1) framework part and (2) implementation part that provides plug-in implementations for the interfaces of the framework. Our experiences of Eclipse Plug-in Development Environment (PDE) revealed that the development of Eclipse plug-ins should be made more fluent. By default, the PDE requires a user to start another instance of the Eclipse workbench before a newly implemented plug-in can be tested. However, starting a workbench is a time and memory consuming task. Therefore, we developed an additional component which makes it possible to develop, debug and run plug-ins that are used in reliability evaluation within one workbench. This component considerably increases usability of the ComponentBee framework and tools.
- From the view point of the tool development, the SMEPP case study was significant in the two ways. For the first time, all tools were used together as an integrated tool chain to support

Table 7
Summary of the integrated tool environment.

Tool	Versions	Extensions	Validation	Remarks
QPE (proprietary, EPL licence)	QPE v 0.5 (1st version)	Relations with quality attributes	A laboratory case study of a weather forecasting domain	Usability has to be enhanced. Especially when working with a huge amount of quality properties
		Eclipse compliant UML tools can utilise defined quality properties	The SMEPP case study	Adding a semi-automatic metric selection would be valuable for the architect
	Tool checks that quality property's value is in a range of the selected metric			
Stylebase For Eclipse (open source, GPL licence)	Q-Stylebase	The initial concept of AKB	A case study of wireless service domain [58]	Increased interest in sharing and exploiting architectural knowledge
	Stylebase for Eclipse v 0.6	C# version for the Tau developer UML tool	A laboratory case study of a distributed service platform [54]	Knowledge consumption is easy, but knowledge maintenance too laborious
	Stylebase for Eclipse v 0.7	Eclipse integration (Java version)	SMEPP case study	The content of AKB brings value, the tool is just an interface
	Stylebase for Eclipse 1.0	Improved QA definition	Feedback from OS users [35]	Tool's best advantage is its integrability and extensibility
	Stylebase for Eclipse v.1.1	Local knowledge repository added Improved usability		
RAP (proprietary, EPL licence)	RAP tool 1.0 1st version	C# version for the Tau developer UML tool	A laboratory case study of a distributed service platform [53]	The first version required a separated simulation model that s is not required anymore. Instead, common UML models are used for evaluation
	RAP tool 2.0 2nd version	Eclipse integration (Java version)	The SMEPP case study	Now it is possible to make evaluation from different usage viewpoints, i.e. usage profiles
	RAP tool 2.5	Usage profiles, integration with ComponentBee, estimated, predicted and measured PoF values in the measurement profile	Industrial personal information repository [40]	Is it possible to evaluate other quality attributes in a similar fashion?
		Producing Markov models semi-automatically Better guidance for the architect in order to estimate initial PoF values		Setting execution order for execution paths
ComponentBee (open source, EPL licence)	RT-tool (1st version)	Prototype for research purposes	A script-based client–server application [43]	A proposal to bring ComponentBee under the Eclipse Technology Project
	ComponentBee v0.9	Improved usability (e.g. possible to run and debug plug-ins in one workbench)	Stylebase for Eclipse [67]	In order to comply with the Eclipse mission, the focus is now on developing extensible framework, rather than tools
	ComponentBee v.1.0	Project divided into two parts: (1) framework, which provides interfaces and (2) exemplary tools which implement these interfaces	The SMEPP case study	

quality aware architecting. In addition, the task was more challenging: the SMEPP middleware platform has hundreds of quality requirements while the software used in previous case studies had less than 30.

5.2. Coverage of the tool chain

The tool chain supports the whole design flow from quality requirements to code evaluation. Downward traceability of quality characteristics works well after quality requirements have been identified and specified. However, transformation from requirements to quality properties seemed to be problematic for the developers. The main reasons seem to be (a) developers' unfamiliarity with quality requirements specification, (b) quality requirements are intertwined with functional requirements, (c) it is difficult to discuss and clarify quality requirements before defining a conceptual architecture or/and the context where the system will be used. The latter is the main reason why the quality requirements specification for a cross-domain architecture is difficult and faulty. However, as soon as the requirements specification has been done in a systematic way, i.e. by utilising quality ontologies, there is no limit to reusing the results. Thus, although the creation of quality ontologies takes time, we believe that it is worth doing. As illustrated, the downward traceability from QA property specifications to models is fully supported by the tool chain. Upward traceability is supported from code to models by a specific UML profile. However, there is still work to be done, e.g. for matching quality properties of architectural models to quality requirements. This could be solved by a new tool that manages traceability links between the artefacts produced in the different phases of the design process.

The development of reliability aware systems is completely supported by the tool chain; from requirements specification to architecture design, evaluation, and testing. The tool suite supports the reliability evaluation on three levels; on system, architecture, and component levels. It is capable of using all the available Probability of Failure (PoF) values (estimated, predicted, and measured) in order to achieve an accurate reliability prediction for a software system.

The three reliability strategies [56], *fault prevention*, *fault removal*, and *fault tolerance*, are supported by the tool chain. Fault prevention uses requirements, designs, and coding techniques and processes, as well as requirements and design reviews, to reduce the number of faults introduced in the first place. Fault

removal uses code inspection and testing to remove faults in the code once it is written Fault tolerance reduces the number of failures that occur by detecting and countering deviations in program execution that may lead to failures. The tool chain including QA ontologies supports all these reliability strategies. The QA ontologies and the QPE, Stylebase and RAP tools together support fault prevention. The QA ontologies, QPE and Stylebase support creation of software architecture for the desired quality goals. The RAP tool, in turn, enables the software architect to evaluate the reliability at design-level before implementation of a software system and thus helps the software architect to develop the software architecture until the desired reliability goals are achieved in it. The ComponentBee supports fault removal by providing a way to identify the failure behaviours of a software system. The tools together support composition of fault tolerant software systems. The QPE, Stylebase and RAP tools facilitate development of more reliable and fault tolerant software architectures whereas the ComponentBee is capable of evaluating how faults affect the dynamic behaviour of the software system.

The quality attributes that are fully or to some extent covered by the tools are: reliability, availability, security, adaptability, maintainability, portability, integrability, and extensibility. We also have a tool for performance monitoring but its integration with other tools still requires adaptation work. We aim to extend the ontology orientation to all execution quality attributes so that a similar approach can be applied to the sub-characteristics of dependability (reliability, availability, security, and safety) and performance. This is extremely important in order to manage several quality attributes and their variations at run-time.

5.3. Comparison to other approaches

Although there exist several methods for modelling quality driven software architectures, to our knowledge there is no other methodology that covers the full life-cycle from the quality requirements specification to measuring quality properties from source code. A short summary of related works is given to illustrate the state of the art in quality and model driven architecture design and quality evaluation.

The NFR (Non-Functional Requirements) framework [14] is a process-oriented approach, in which quality requirements are treated as soft-goals to be achieved. The soft-goals are derived from the stakeholders' needs and used as guidance while considering design alternatives, analysing tradeoffs, and rationalising various design decisions. A soft-goal interdependency graph is used to support the goal oriented process of architecture design. We have exploited the ideas of NFR but our methodology deals with traceability of both evolution and execution quality attributes from requirements specification to architectural models and code and provides means of modelling, evaluating and testing quality attributes.

CBSP (Component Bus System Property) [30] defines five steps starting from taking a requirement under consideration and finishing with making tradeoffs regarding architectural elements and styles. Each requirement is assessed for its relevance to the system architecture components and connectors (buses), to the topology of the system or a particular sub-system, and to their properties. The intermediate CBSP model is used as a bridge while refining and transforming requirements to components and connectors. Although the coverage of CBSP is pretty much the same as our methodology, our approach is a systematic approach that keeps focus on knowledge reuse, model based evaluation and model-based testing and provides appropriate tooling for each engineering phase.

Architecture Based Design (ABD) method [2] is a well-known quality driven method for designing the software architecture for a long-lived system. The method addresses functional, quality, and business requirements and their mappings to the conceptual architecture. ABD provides a series of steps for designing the conceptual software architecture and ends when decisions on concrete components are to be made. In addition, the method provides a collection of software templates that constrain the implementation of components of different types. Even though the ABD method has been developed further into a new method called the Attribute Driven Design method, ADD [3], it still does not provide more than a coarse grained high level, i.e. conceptual architecture as an output. Also the support for product line architecture design in the ABD and in the ADD is mentioned but immature. Moreover, no modelling tool is provided.

QASAR (Quality Attribute oriented Software ArchitectuRe design method) [6] is another model driven architecture design method that addresses quality attributes in software architecture design. QASAR is consisting of two iterative processes: the inner iteration includes the activities of software architecture design, assessment and transformation to quality requirements, whereas the outer iteration refers to a requirements selection process to be performed within the inner iteration. The method starts from functionality and adapts functionality to quality requirements. Thus, in this method quality requirements are not considered a driving force in architecture development.

Regarding the abstractions, viewpoints and languages for modelling the conclusion is that the existing methods offer good coverage but none of them give full support, for e.g. quality requirements specification and their traceability throughout the life-cycle of the software development. Some commercial tools, like IBM Rational DOORS, have established links between certain commercial modelling tools but to our knowledge no open source tool supports the link between requirements engineering and software architecting. However, some open source transformation tools (e.g. ATL Development Tools for Eclipse) have demonstrated a great degree of maturity and offer flexible transformation patterns. In summary, Eclipse based modelling tools offer a higher degree of integration with transformation tools than other similar open or commercial tool environments. Moreover, commercial tools have shown to be complete Model Driven Engineering frameworks by themselves, making any kind of modifications in the model to code transformation difficult.

There exist several quality evaluation methods that address different quality attributes, goals, stakeholders (i.e. method users), and therefore they also provide different evaluation results. The differences, for example, between performance and reliability evaluation methods are the maturity of methods and tools provided for evaluation steps. The main reason is that performance evaluation has been studied for tens of years but reliability evaluation at the architecture level has been addressed only for a few years. The main shortcomings of the existing methods are: precision of the evaluation results, lack of tooling and missing references from industrial case studies.

Most evaluation methods are used on the component, sub-system and/or system levels. However, one factor seems to be common for most evaluation methods: UML is a standard modelling language and supporting tools are based on it. Although most evaluation methods require a specific set of architectural models as input, the use of MDA and specific profiles for quality attributes is not supported.

5.4. Future research items

This last experiment, where the SMEPP middleware was evaluated with the tool chain, revealed some topics that need further research:

QPE should facilitate metric selection especially when we have a lot of quality metrics available. One possibility is that QPE filters metrics, and the architect could select a metric from a filtered metric group. Probably, this kind of feature can be implemented by utilising existing filtering and data mining techniques.

The Stylebase for Eclipse tool is only an interface to the Architectural Knowledge Base (AKB), and therefore, the real value of the tool depends on the contents of the AKB. Currently, our knowledge base contains mostly generic solutions such as architecture patterns from Buschmann [11]. Next, we plan to gradually accumulate a large knowledge base of domain-specific patterns and other specialised solutions. However, the knowledge maintenance features of Stylebase for Eclipse must first be improved.

Extending RAP for predicting other execution quality attributes. The defined usage profile is not quality attribute dependent and it could be utilised directly for a new prediction algorithms. Therefore, research of additional prediction methods based on architectural models is required. However, before that there has to be suitable metrics available for these quality attributes (cf. Probability of Failure PoF).

Currently, ComponentBee supports reliability testing. In the future, the tool could be extended and made capable to test other execution qualities, such as performance and availability. It should be quite straightforward to take the technique for calculating reliability values and use them to calculate other quality values. The essential concept would remain the same: (1) test models are constructed (2) dynamic behaviour is logged, (3) behaviour patterns are recognised from the raw log (4) and quality values are calculated.

In the future, we hope to validate the tool chain in a genuine industrial setting (e.g. oversee its deployment in a software company). However, further effort is required before this is feasible. For example, quality ontologies for all execution qualities are to be defined. Moreover, the usability of the approach would be much better if backward traceability is also supported and if all models needed for quality evaluation could be automatically transformed from the architecture models. Nowadays, only reliability evaluation models are automatically extracted from the architectural models.

6. Conclusion

In this paper, we introduced a quality aware software architecting approach based on the novel techniques in the knowledge engineering and model driven software engineering areas. The approach supports design time quality management and covers the whole development life cycle of software systems starting from quality requirements specification, selecting measuring techniques for quality properties, representing quality properties in architectural models, and finally, evaluating how well the quality requirements are met. Furthermore, it provides supporting methods, techniques, and an integrated tool chain that is based on the Eclipse platform.

The quality aware modelling approach has three main phases; modelling quality requirements, representing quality in architectural models, and model based quality evaluation. In the modelling quality requirements phase, quality knowledge is defined as quality attribute ontologies. This knowledge is exploited in the quality requirements specification activity for producing reusable quality properties. Architectural knowledge is stored as model artefacts; styles and patterns, and used for representing quality in architecture. Architectural knowledge is also used while qualitative evaluation methods are applied to evaluating evolution qualities. Meanwhile, execution qualities are evaluated from the architectural models annotated with quality properties by using reusable usage profiles. Thus, knowledge engineering is used for improving reuse of knowledge of quality attributes and architectures. Reusable models make architecting efficient and of high quality.

Although our approach has been proved to work in practice, we have also identified several research topics, which need further examination. One of the most important topics is that all quality attributes related to execution qualities should be fully supported. Thereafter, we have a chance to take a step forward and develop mechanisms and algorithms for managing quality at run-time.

Acknowledgements

The publication of this paper has been supported by the following research projects: EU-SMEPP (EU-FP6-IST 0333563), ITEA-COSI, ITEA-CAM4HOME and ARTEMIS-SOFIA. The first mentioned project is mainly funded by the European Commission. The three others are funded by the National Technology Agency (Tekes) and VTT Technical Research Centre of Finland.

References

[1] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, D. Perry, Architectural knowledge and rationale: issues, trends, challenges, SIGSOFT Softw. Eng. Notes 32 (4) (2007) 41–46 (July).
[2] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, F. Peruzzi, The Architecture based Design Method, CMU/SEI, Technical Report 2000-TR-001, 2000.
[3] F. Bachmann, L. Bass, Introduction to the attribute driven design method, in: 23rd International Conference on Software Engineering (ICSE'01), 0745, 2001.
[4] D. Brickley, R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C, 2004. <http://www.w3.org/TR/rdf-schema/>.
[5] A. Barstow, J. Hendler, M. Skall, et al. OWL Web Ontology Language for Services, W3C, 2004. <http://www.w3.org/Submission/2004/07/>.
[6] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach, Addison-Wesley, Harlow, 2000.
[7] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, B. Selic, An MDA Manifesto, in: MDA Journal: Model Driven Architecture Straight from the Masters, Meghan-Kiffer Press, Tampa, FL, USA, 2005.
[8] G. Booch, I. Jacobson, J. Rumbaugh, Unified Modeling Language for Object Oriented Development, Rational Software Corporation, 1996.
[9] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0. W3C, 1999. <http://www.w3.org/TR/REC-xml/>.
[10] J. Bruck, Defining Generics with UML Templates, IBM, 2007. <http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Defining_Generics_with_UML_Templates/article.html>.
[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sammerlad, M. Stal, Pattern Oriented Software Architecture: A System of Patterns, John Wiley & Sons Ltd., Chichester, 1996.
[12] S.W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, N. Hu, Software architecture-based adaptation for grid computing, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), 2002.
[13] R.C. Cheung, A user-oriented software reliability model. Software engineering, IEEE Transactions on Software Engineering 6 (2) (1980) 118–125.
[14] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulus, Non-functional Requirements in Software Engineering, Kluwer Academic Publishers, Boston, 2000.
[15] L. Dobrica, E. Niemelä, A Survey on Software Architecture Analysis Methods, IEEE Transactions on Software Engineering 28 (7) (2002) 638–653.
[16] G. Dobson, R. Lock, I. Sommerville, QoSOnt: a QoS ontology for service-centric systems, in: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005.
[17] T. Edgington, B. Choi, K. Henson, T. Raghu, A. Vinze, Adopting ontology to facilitate knowledge sharing, Communications of ACM 47 (11) (2004) 85–90.
[18] J.R. Evans, W.M. Lindsay, The Management and Control of Quality, South Western College Publishing, 1999.
[19] A. Evesti, E. Niemelä, Quality-oriented architecting environment for quality variability, in: Proceedings of International Conference and Software and Systems Engineering and their Applications, ICSSEA, 3–6 December, 2007.
[20] A. Evesti, E. Niemelä, K. Henttonen, M. Palviainen, A tool chain for quality-driven software architecting, in: Proceedings of the 12th International Software Product Line Conference (SPLC '08), 2008.
[21] R. Farenhorst, P.Lago, H. van Vliet, Prerequisites for successful architectural knowledge sharing, in: Proceedings of the 18th Australian Software Engineering Conference (ASWEC 2007), 2007.
[22] R. Farenhost, P. Lago, H. Vlient, Effective tool support for architectural knowledge sharing, Lecture Notes in Computer Science 4758/2007, Springer-Verlag, Berlin, Heidelberg, 2007.
[23] U. Farooq, C. Lam, H. Li, Towards automated test sequence generation, in: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC 2008), 2008.
[24] M. Fernandez-Lopez, A. Gomez-Perez, N. Juristo, METHONTOLOGY: from ontological art towards ontological engineering, in: Proceeding of Spring Symposium on Ontological Engineering of AAAI, Stanford University, California, 1997.

[25] FIPA, FIPA Quality of Service Ontology Specification 2009/02/11, Foundation for Intelligent Physical Agents, 2002. <http://www.fipa.org/specs/fipa00094/XC00094.html>.
[26] R. France, B. Rumpe, Model-driven development of complex software: a research roadmap, Future of Software Engineering (2007) (FOSE '07).
[27] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Desing Patterns. Elements of Reusable Object-oriented Software, Addison-Wesley, Boston, 1994.
[28] L.M. Garshol, BNF and EBNF: what are they and how do they work? 2002. <http://www.garshol.priv.no/download/text/bnf.html>.
[29] H. Glaser, A. Jaffri, I. Millard, B. Rodriguez, ReSIST Ontology, 2006. <http://users.ecs.soton.ac.uk/aoj04r/resist.owl>.
[30] P. Gruenbacker, A. Egyed, N. Medvidovic, Reconciling software requirements and architectures with intermediate models, Software and Systems Modeling 3 (3) (2003) 235–253.
[31] W. Hasselbring, R. Reussner, Toward Trustworthy Software Systems, Computer 39 (4) (2006) 91.
[32] P. Hayes, RDF Semantics, W3C, 2004. <http://www.w3.org/TR/rdf-mt/>.
[33] R. Heckel, M. Lohmann, Towards model-driven testing, Electronic Notes in Theoretical Computer Science 82 (6) (2003) 33–43.
[34] K. Henttonen, Stylebase for Eclipse, An open source tool to support the modeling of quality-driven software architecture, Research Note 2387, Espoo: VTT Technical Research Centre of Finland, 2007. <http://www.vtt.fi/inf/pdf/tiedotteet/2007/T2387.pdf>.
[35] K. Henttonen, M. Matinlassi, Contributing to Eclipse: a case study, in: Proceedings of the 2007 Conference on Software Engineering (SE2007), Hamburg, Germany, 29–30 March, 2007.
[36] K. Henttonen, M. Matinlassi, Open source based tools for sharing and reuse of software architectural knowledge. Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) & 3rd European Conference on Software Architecture (ECSA), Cambridge, UK, 14–17 Sept. 2009. IEEE, pp. 41–50.
[37] K. Henttonen, M. Matinlassi, E. Niemelä, T. Kanstrén, Integrability and extensibility evaluation from software architectural models – a case study, The Open Software Engineering Journal 1 (1) (2007) 1–20 (Bentham Science Publishers, Sharjah).
[38] IEEE, IEEE 982.1, In IEEE Standard Dictionary of Measures of the Software Aspects of Dependability, Institute of Electrical and Electronics Engineers, ISBN: 0738148466, 2005.
[39] A. Immonen, A method for predicting reliability and availability at the architectural level, in: T. Käkölä, J.C. Dueñas (Eds.), Software Product-lines – Research Issues in Engineering and Management, Springer-Verlag, Berlin, 2006, pp. 373–422.
[40] A. Immonen, A. Evesti, Validation of the reliability analysis method ant tool, in: Proceedings of the 12th International Software Product Line Conference (SPLC '08), vol. 2, 2008.
[41] A. Immonen, E. Niemelä, Survey of reliability and availability prediction methods from the viewpoint of software architecture, Software and Systems Modelling 7 (1) (2008) 49–65.
[42] A. Immonen, A. Niskanen, A tool for reliability and availability prediction, in Proceedings of the 31st Euromicro Conference on Software Engineering and Advanced Applications, Porto, IEEE Computer Society, 30 August–3 September, 2005.
[43] A. Immonen, M. Palviainen, Trustworthiness evaluation and testing of open source components, in: Seventh International Conference on Quality Software (QSIC'07), Portland, Oregon, USA, October 11–12, 2007.
[44] ISO/IEC, 9126:2-4, Software Engineering, Product Quality, Parts 4–6, 2003 <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749>.
[45] ISO/IEC, 9126-1, Software Engineering, Product Quality, Part 1: Quality Model, 2001.
[46] J. Kalaoja, T. Paaso, J. Toivonen, P. Plaza, J. Marina, J. Serrano, S. Kraxberger, D. Garrido, R. Roman, M. Diaz, D3.2 Conceptual Architecture of Secure EP2P Middleware, 2008. <http://www.smepp.org/DownloadsReview/D3_2_Final_Year2.pdf>.
[47] J. Kalaoja, T. Paaso, M. Rodrigues, D. Garrido, A. Rayna, P. Merino, A. Recio, F. Benigni, R. Popescu, J. Serrano, J. Marina, D3.3 Concrete Architecture of Secure EP2P Middleware, 2008. <http://www.smepp.org/DownloadsReview/D3_3_Final_Year2.pdf>.
[48] J. Kantorovitch, E. Niemelä, Service description ontologies, in: Mehdi Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology, second ed., vol. 7, Published under the imprint Information Science Reference (formerly Idea Group Reference), 2008, pp. 3445–3451.
[49] S. Kelly, J.Tolvanen, Domain-specific Modelling: Enabling Full Code Generation, ISBN: 978-0-470-03666-2, 2008.
[50] M. Matinlassi, E. Niemelä, L. Dobrica, Quality-driven Architecture Design and Quality Analysis Method. A Revolutionary Initiation Approach to a Product Line Architecture, VTT Electronics, Espoo, Finland, 2002.
[51] D. McGuinness, F. van Harmelen, OWL Web Ontology Language Overview, W3C, 2004. <http://www.w3.org/TR/owl-features/>.
[52] S.J. Mellor, S. Kendal, A. Uhl, D. Weise, MDA Distilled, Addison-Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.
[53] J. Merilinna, A Tool for Quality-driven Architecture Model Transformation, VTT Technical Research Centre of Finland, Espoo, 2005.
[54] J. Merilinna, E. Niemelä, A stylebase as a tool for modeling of quality-driven software architecture, in: Proceedings of the Estonian Academy of Sciences, Special Issue on Programming Languages and Software Tools, vol. 11(4), Tallinn University of Technology, Tallinn University, Estonian Agricultural University, 2005.
[55] J. Miller, J. Johansson, MDA Guide, Object Management Group, 2003. <http://www.omg.org/docs/omg/03-06-01.pdf>.
[56] J.D. Musa, Software-reliability-engineered testing, Computer (1996) 61–68.
[57] E. Niemelä, A. Evesti, P. Savolainen, Modeling quality attribute variability, in: Third International Conference on Evaluation of Novel Approaches of Software Engineering, ENASE 2008, Funchal, Madeira, Portugal, 4–7 May, 2008.
[58] E. Niemelä, J. Kalaoja, P. Lago, Toward an architectural knowledge base for wireless service engineering, IEEE Transactions on Software Engineering 31 (5) (2005). 361-362-379.
[59] E. Niemelä, M. Matinlassi, Quality evaluation by QADA, in: A half-day tutorial in the 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005, Pittsburg, Pennsylvania, USA, 6–9 November, 2005.
[60] L. O'Brien, P. Merson, L. Bass. Quality Attributes for Service-oriented Architectures, in: Proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA '07), 20–26 May, 2007.
[61] Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, 2008. <http://www.omg.org/spec/QVT/1.0/>.
[62] Object Management Group, UML Superstructure Specification 2.0, 2005 <http://www.omg.org/spec/UML/2.0/>.
[63] Object Management Group, A Proposal for an MDA Foundation Model, 2005 <http://www.omg.org/docs/ormsc/05-04-01.pdf>.
[64] Object Management Group, CORBA Component Model, v.4.0, 2006. <http://www.omg.org/docs/formal/06-04-01.pdf>.
[65] Object Management Group, MARTE Specification, 2007. <http://www.omgmarte.org/Specification.htm>.
[66] D. Pakkala, P. Pääkkönen, M. Sihvonen, A generic communication middleware architecture for distributed application and service messaging, in: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005.
[67] M. Palviainen, A model-based method for dynamic behaviour and reliability evaluation of multithreaded Java programs. in: Submitted to the Third International Conference on Software Testing, Verification and Validation (ICST 2010), Paris, France, 2010.
[68] H. Robinson, Graph theory techniques in model-based testing, in: In the International Conference on Testing Computer Software, 1999.
[69] RUP, Rational Unified Process: Best Practices for Software Development Teams, 2009. <http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html>.
[70] D. Rubel, The Heart of Eclipse, ACM Queue 4 (6) (2006) 36–44.
[71] P. Savolainen, E. Niemelä, R. Savola, A taxonomy of information security for service-centric systems, in: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, 20075-12, 2007.
[72] B. Selic, A Definition of MDA, Brest, Brittany, France, Presentation in the Second Summer School "MDA for Embedded Systems", 6–10 September, 2004.
[73] SCRUM, What is Scrum? 2009. <http://www.controlchaos.com/about/>.
[74] M. Tian, A. Gramm, H. Ritter, J. Schiller, Efficient selection and monitoring of QoS-Aware Web services with the WS-QoS framework, in Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, 2004.
[75] P. Tarvainen, Adaptability evaluation at software architecture level, The Open Software Engineering Journal 2 (1) (2008) 1–30 (Bentham Science Publishers, Sharjah).
[76] P. Tarvainen, Adaptability evaluation of software architectures; a case study, in: Proceedings of the 31st Annual International Computer Software and Applications Conference, COMPSAC, 2007, pp. 2579–2586.
[77] M. Utting, Position Paper: Model-based Testing, in: Proceedings of the Verified Software: Theories, Tools, Experiments (VSTTE) Conference, 2005.
[78] M. Utting, A. Pretschner, B. Legeard, A Taxonomy of Model-based Testing, Department of Computer Science, The University of Waikato, Hamilton, New Zealand, 2005.
[79] C. Vairo, M. Albano, S. Chessa, A secure middleware for wireless sensor applications, in: The Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Trinity College Dublin, Ireland, 21–25 July, 2008.
[80] C. Zhou, L. Chia, B. Lee, DAML-QoS Ontology for Web services, in: Proceeding of the International Conference on Web Services (ICW2004), 2004, pp. 472–479.
[81] C. Zhou, L. Chia, B. Lee, QoS measurement issues with DAML-QoS ontology, in: IEEE International Conference on Business Engineering, ICEBE, 2005.
[82] J. Zhou, Knowledge dichotomy and semantic knowledge management, in: 1st IFIP WG 12.5 Working Conference on Industrial Applications of Semantic Web, Jyvaskyla, Finland, 2005.
[83] J. Zhou, E. Niemelä, A. Evesti, A. Immonen, P. Savolainen, OntoArch approach for reliability-aware software architecture development, in: Proceedings of QACOS2008, IEEE Computer Society, ISBN: 978-0-7695-3262-1, 2008.
[84] J. Zhou, E. Niemelä, P. Savolainen, An integrated QoS-aware service development and management framework, in: Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA), Mumbai, India, 6–9 January, 2007.

PUBLICATION V

The design, instantiation, and usage of information security measuring ontology

In: Proceedings of the Second International Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS), Budapest, Hungary, 17–22 April 2011. Pp. 1–9.

The Design, Instantiation, and Usage of Information Security Measuring Ontology

Antti Evesti, Reijo Savola, Eila Ovaska, Jarkko Kuusijärvi
VTT Technical Research Centre of Finland
Oulu, Finland
e-mail: antti.evesti@vtt.fi, reijo.savola@vtt.fi, eila.ovaska@vtt.fi, jarkko.kuusijarvi@vtt.fi

***Abstract*—Measuring security is a complex task and requires a great deal of knowledge. Managing this knowledge and presenting it in a universal way is challenging. This paper describes the Information Security Measuring Ontology (ISMO) for measuring information security. The ontology combines existing measuring and security ontologies and instantiates it through example measures. The ontology provides a solid way to present security measures for software designers and adaptable applications. The software designer can utilise the ontology to provide an application with security measuring capability. Moreover, the adaptable application searches for measures from the ontology, in order to measure a security level in the current run-time situation. The case example illustrates the design and run-time usage of the ontology. The experiment proved that the ontology facilitates the software designer's work, when implementing security measures for applications that are able to retrieve measures from the ontology at run-time.**

Keywords-adaptation; run-time; quality; measure; security metric; software

I. INTRODUCTION

Software applications running on devices and systems may face needs for changes due to alterations happening in their execution environments or intended usages. These changes may have a considerable effect on the security requirements of the software system. Moreover, emerging security threats and vulnerabilities may affect the achieved security level. However, the software system is required to achieve a desired security level in these changing circumstances [1]. Therefore, the software has to be able to observe the security level at run-time, measure the fulfilment of the security requirements, and adapt itself accordingly. However, measuring the security level at run-time requires the correct measures and measurement techniques for each situation. Defining the measures and the measuring techniques is a time consuming task and requires the use of experts from different domains. Thus, it is important to present the defined measures in a universal and reusable form. In addition, problems concerning how to present these measures, the measuring techniques, and their mutual relationships have to be solved in a way that facilitates run-time security measuring. Ontologies provide a possibility to manage this knowledge, making it possible to describe different security requirements and ways to measure the fulfilment of these requirements.

Ontologies are utilised in [2] to achieve the required quality of the software at a design-time. Thus, it is reasonable to utilise ontologies as a knowledge base for quality management at the run-time. Furthermore, the work in [3] presents the architecture for developing software applications with security adaptation capabilities – the presented approach assumes that the knowledge required for security monitoring and adaptation is available from ontologies.

In this work, we will present a novel ontology for the run-time security measuring – called Information Security Measuring Ontology (ISMO). ISMO combines a terminology from a software measurement area in general and the security related terminology. In addition, a few security measurements are added to the ontology in order to instantiate it. The novelty of our work comes from this combination – based on our current knowledge, there isn't any other ontology which describe security measuring in a run-time applicable way. The content of ISMO can be enhanced after the software application has been delivered. Hence, the measuring process is based on the up-to-date specifications of security measures. The purpose of this new ontology is to make it possible for software applications to utilise security measures during run-time in changing environments. Therefore, it is possible for the application to measure the fulfilment of its security requirements and adapt the used security mechanisms if the required security is not met. In other words, the measuring acts as a trigger for the adaptation. However, to achieve software applications with a measuring capability, the developer must have implemented a set of measuring techniques as a part of it. Thus, ISMO also provides input for developers – presenting what measuring techniques are to be implemented and how.

The remainder of the paper is organised as follows: Section 2 provides background information; Section 3 presents an overview of the combined ontology and mentions some security measurements. Section 4 instantiates the defined ontology. Section 5 explains how the ontology is utilised. A case example is presented in Section 6. Finally, a discussion and conclusions close the paper.

 ISBN: 978-1-61208-005-5

II. BACKGROUND

ISO/IEC defines security in [4] as follows: "The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them." Furthermore, in some sources security is thought to be a composition of confidentiality, integrity and availability [5, 6]. In [7], these security sub-attributes are called security goals.

Zhou [8] defines ontology as a shared knowledge standard or knowledge model, defining primitive concepts, relations, rules and their instances, which comprise topic knowledge. It can be used for capturing, structuring and enlarging explicit and tacit topic knowledge across people, organizations, and computer and software systems.

Blanco et al. [9] lists several security ontologies in their work. In addition, our earlier work [10] also compares a number of security ontologies, particularly those that are applicable for run-time usage. It is noticed in [10] that security ontologies for run-time usage exist – especially for service discovery and matchmaking, for example, ontologies from Denker et al. [11] and Kim et al. [12]. In addition, security ontologies which concentrate on software design and implementation phases also exist, e.g., works from Savolainen et al. [13] and Tsoumas et al. [14]. From these ontologies, only the work from Savolainen et al. [13] takes measurements into account, by presenting a high level classification for different security measures. However, the most extensive information security ontology at the moment is the one proposed by Herzog et al. [7], called an ontology of information security – abbreviated as (OIS) in this work. The OIS is intended to provide a general vocabulary or an extensible dictionary of information security. It is applicable at design and run-time alike, and it contains more concepts than all the above mentioned security ontologies altogether. Thus, this ontology provides a sound starting point for defining the concepts of ISMO.

The OIS does not contain concepts for describing measures. Therefore, the Software Measurement Ontology (SMO) [15] is utilised for measurement definitions. The SMO presents the generic measurement terminology related to software measurements. In other words, ontology is quality attribute independent. The SMO collects and aligns terminology from several standards of software engineering, software quality metrics, and general metrology. It is important to notice that the SMO uses a term measure instead of metric. Thus, the measure term will be used in this work. The SMO divides measures into three sub-classes: namely a base measure, derived measure, and indicator – all of which inherit the same relationships from other concepts. The base measure is an independent 'raw' measure. A derived measure is a combination of other derived measures and / or base measures. Finally, an indicator can be a combination of all of these three types of measures. The complexity of these measures increases when moving from base measures to derived measures and further on to indicators. In literature, base measures and derived measures are also called direct and indirect measures; however we will follow the terminology defined by the SMO.

Hence, this work draws mappings between the OIS and SMO, instead of defining a new ontology from scratch. This is considered to be reasonable since a remarkable effort has been invested into these existing ontologies and both are scientifically reviewed and accepted. In addition, the reuse of existing ontologies is suggested in [16] as one potential approach for ontology development.

III. THE DESIGN OF INFORMATION SECURITY MEASURING ONTOLOGY

This section describes how the combined ontology ISMO is designed. SMO [15] contains 20 generic measurement related concepts and their relationships. Thus, security measures will be used to instantiate ontology for security measuring purposes – creating base measures, derived measures, and indicators. On the other hand, OIS [7] contains concepts related to threats, assets, countermeasures, security goals, and the relationships between those concepts. In addition, the OIS describes a couple of vulnerabilities and how these act as enablers for threats. The OIS already contains some of these concepts as an instantiated form, such as the security goals of authentication, integrity, etc.

The purpose of combining these two ontologies is to achieve an ontology that makes it possible to measure the fulfilment of security requirements, i.e., security goals and levels. In other words, the purpose is to enable an operational security correctness measurement, as called in [17]. Therefore, the requirements are described by a means of vocabulary from the OIS. The requirements fulfilment is measured with indicators – which combine several measures – defined in the SMO. The security measures, i.e., indicators, are different for each security goal, e.g., a level of authentication or non-repudiation is measured with different measures. However, these measures can utilise the same base measures. On the other hand, the same security goal can be achieved with different countermeasures, which in turn might require their own measures. For instance, the authentication level, which is achieved, is measured in a different way when a security token is used instead of a password authentication. Hence, there are only a few concept-to-concept mappings between these two ontologies, but additional mappings appear when the measures are instantiated. By using a terminology from ontologies, a mapping refers to the property between the concepts. Adding mappings for instantiated measures requires domain expertise, i.e., the capability to recognise applicable measuring techniques for a particular security goal and related mechanisms. Furthermore, the mapping requires a capability to recognise threats which affect to the particular security goal and/or mechanism. Mappings at the instantiation phase are described more detail in Section IV.

Fig. 1 shows an overview of the combining process. Firstly, the mappings between the concepts are drawn. Secondly, security measure instances are added and related mappings for each measure are defined. Thus, the SMO is used as a guideline as how to define the instances of security measures.

 ISBN: 978-1-61208-005-5

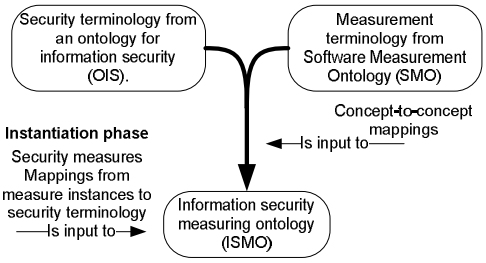


Figure 1. An overview of the combining process.

TABLE I shows mapping properties between concepts from the SMO and OIS. These are mappings made from concept to concept. Each *SecurityGoal* has an *Indicator* – intended for measuring the fulfilment of the goal. The SMO uses the term *QualityModel* for defining measurable concepts. *QualityModel* is a quality attribute dependent, i.e., security in this case. Thus, the *QualityModel* concept is related to *SecurityGoal*. The *MeasurableConcept* from the SMO is also mapped to the *SecurityGoal,* through the means of the *isDefinedFor* property.

In the SMO, *MeasurableConcept* relates to *Attribute*, meaning a characteristic that will be measured. Thus, *Attribute* can relate to countermeasures, threats, or assets, depending on the measure which is used. *hasMeasurableAttribute* is optional, meaning that mappings to the attributes are made during the phase when the measures are instantiated.

TABLE I. MAPPING PROPERTIES BETWEEN SMO AND OIS

Concept from OIS	Mapping property (direction)	Concept from SMO
SecurityGoal	hasIndicator (->)	Indicator
SecurityGoal	isRelatedTo (<-)	QualityModel
SecurityGoal	isDefinedFor (<-)	MeasurableConcept
Countermeasure, Threat, Asset	hasMeasurableAtt-ribute (->) (optional)	Attribute

A. Security Measures

The overall security level of the product can be represented by a combination of relevant security attribute measures. However, it is not possible to cover all security measures in this work. Consequently, we will concentrate on user authentication, and thus, the ISMO is instantiated by these measures.

In [18] measures for various security goals (e.g., authentication, integrity, etc.) are defined by using a decomposition approach introduced by Wang et al. in [19]. Authentication can be decomposed into five components – called BMCs (Basic Measurable Components) – as follows: Authentication Identity Uniqueness (AIU), Authentication Identity Structure (AIS), Authentication Identity Integrity (AII), Authentication Mechanism Reliability (AMR), and Authentication Mechanism Integrity (AMI). Savola and Abie [18] define equations for these BMCs, and in addition, the equation for combining Authentication Strength (AS) from these BMCs. AS is an aggregated user-dependent measure that can be utilized in authentication and authorization.

The user-dependent AS results can be combined into a system-level AS, which can be utilized in run-time adaptive security decision-making [18]. When considering a software application that measures its security level at run-time, AIS, AII, and AMR are particularly applicable. In other words, an application cannot measure AIU and AMI, as the information which is required for these measures is only available at the server side where the application will be authenticated. Thus, in this work, the measures for the AIS will be used as example measures.

To measure AIS, we utilise a measure intended for situations where the authentication is based on a password – called the structure of the password. It is commonly understood that the structure of a password, i.e., the length and variation of the symbols, affects the achievable authentication strength. Therefore, we divide passwords into groups such as: i) a PIN code containing four numbers, ii) a password containing 5-9 lower case characters, and iii) a password containing over 10 ASCII symbols. Intuitively, *group i* provides the worst authentication level, *group ii* offers an increased authentication level, and *group iii* is the best alternative.

Another measure that we utilise for password based authentication is the age of the password, i.e., how long the same password has been used. This measure can be used for two different purposes. Firstly, to measure the security policy fulfilment, e.g., a policy can define that the password has to be changed every three months. Secondly, the measure can be utilised as a factor of measuring the authentication strength by utilising more complex analysis models. The age of the password is also mapped to the AIS from BMCs.

These two measures are simple to understand, and thus, provide a good starting point for instantiating ISMO. The graphs in Fig. 2 illustrate how the password strength is affected by the structure and age of the password. These are however merely examples and we do not claim that these affects are linear.

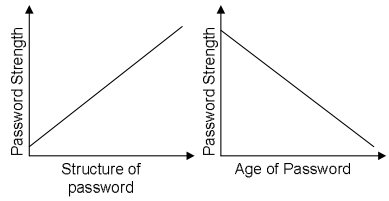


Figure 2. Conceptual correlation graphs for the authentication measures.

IV. THE INSTANTIATION OF INFORMATION SECURITY MEASURING ONTOLOGY

In this section, the authentication related measures are instantiated as a part of the ISMO and the required mappings are added. Fig. 3, Fig. 4, and Fig. 5 present the instantiated ontology – rectangles depict the concepts from SMO and ellipses refer to concepts from OIS. The name of each

 ISBN: 978-1-61208-005-5

concept is presented in the figures and separated from the instance name by a colon, i.e., ConceptName : InstanceName. The property mappings between these two ontologies are presented in bold fonts. For reasons of clarity, the instantiated ontology is presented in three separated figures. Consequently, some concepts may appear in each figure, but from a different viewpoint, i.e., presenting different properties.

The SMO contains the concept *MeasurableConcept*, which corresponds conceptually to the BMCs, which are defined in [18]. Thus, AIS BMC is instantiated in the ontology as a *MeasurableConcept*. The concept *QualityModel* refers to security goals in this work. Hence, there is a mapping property from the *QualityModel* concept to Security goals (authentication, confidentiality, etc.), as mentioned in Section 3.

The *PasswordAge* measure (Fig. 3) is the first measure instance which is added to the ISMO. In the SMO, measures are defined for attributes and these attributes are related to the measurable concept. AIS is an instance of the measurable concept and *PasswordAge* is one of the related attributes. This attribute is measured through the means of an instantiated derived measure, called *UsageTime*. Hence, the derived measure is not purely security related, and can also be applied for other attributes, e.g., the usage time of the CPU in performance measurements. The derived measure *UsageTime* is calculated with a measurement function – defining that *UsageTime* is the current date minus the starting date. The calculation of the value for this measurement function requires that a base measure instance called *Date* is used. *Date* is a base measure, meaning that it is not dependant on other measures and its value is measured by the measurement method. The measurement method for the *Date* measure is simple: taking the date value from the system clock. Defining *UsageTime* as a derived measure may seem like an overestimation. However, detailed definitions are required in order to achieve a measuring ontology that supports run-time security measuring.

The second measure – presented in Fig. 4 – is connected to the AIS via an attribute called *PasswordStructure*. The attribute is measured with an instantiated indicator called *PasswordType*. Indicators are calculated using an analysis model. In this context, the analysis model is a set of rules, which can be thought as if-then-else statements. We have decided to use statements which are very close to the natural English language, so that the analysis models could be updated without an extensive knowledge on programming. The statements of the analysis model can be updated later on to, e.g., the standard SPARQL [20] queries. The analysis model itself is saved as a string literal, so it can be easily changed into a SPARQL statement. For simplicity, the following analysis model is defined for the *PasswordType*:

- |Length < 5 AND OnlyNumbers := PINCode|
- |Length >= 5 AND Length <= 9 AND OnlyAlphabets := NormalPassword|
- |Length > 9 AND Length < 12 AND NumberOfDifferentSymbols >= 3 := GoodPassword|
- |ELSE := WeakPassword|

Thus, four base measures are used in this analysis model, i.e., *OnlyNumbers*, *OnlyAlphabets*, *Length* and *NumberOfDifferenSymbols*. These are measured using appropriate measurement methods, respectively (methods for *OnlyNumbers* and *OnlyAlphabets* are omitted from Fig. 4. In addition, these base measures are also connected to the AIS via appropriate attributes. For reasons of clarity, these are not presented in Fig. 4; however, TABLE II also lists these attributes.

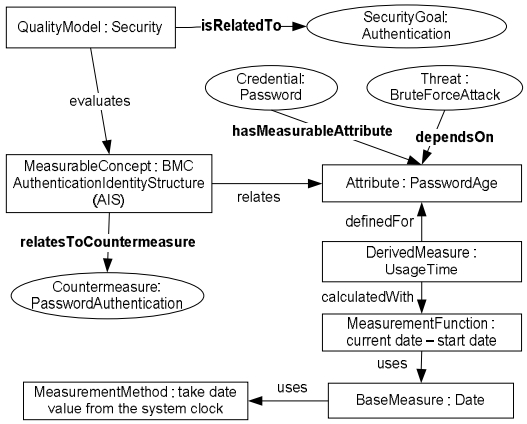


Figure 3. The age of the password.

The above mentioned attributes *PasswordStructure* and *PasswordAge* are mapped to the *BruteForceAttack* threat from the OIS. Thus, these are possible extension points in the future, in so far as risk related measures are added to ISMO. The risk measures are applied for run-time usage in [21].

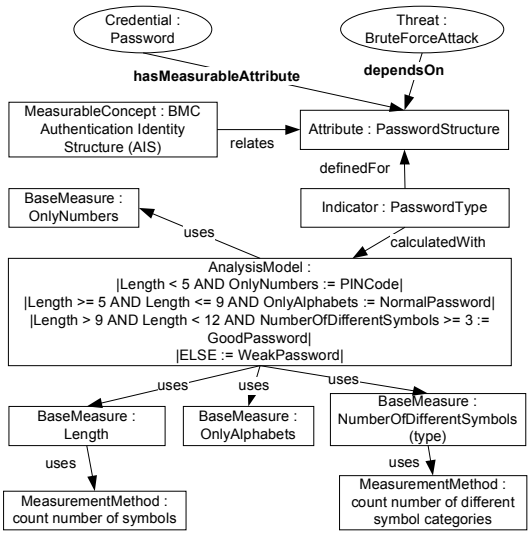


Figure 4. The structure of the password.

 ISBN: 978-1-61208-005-5

The final measure instantiated into the ontology is *AuthenticationLevel* (Fig. 5), which is an instance of the indicator concept – intended to combine the above described measures. The *AuthenticationLevel* is calculated with an analysis model in a similar manner as described for the *PasswordType* above. The analysis model is as follows:

- |PasswordType == PINCode := Level1|
- |PasswordType == NormalPassword AND UsageTime >= 180 AND UsageTime < 365 := Level2|
- |(PasswordType == GoodPassword AND UsageTime < 180) OR (PasswordType == NormalPassword AND UsageTime < 90 := Level3)|
- |ELSE := Level1|

The analysis model for the *AuthenticationLevel* uses the results from the *PasswordType* indicator and the *UsageTime* derived measure. Therefore, the calculation of the authentication level, according to this analysis mode, requires the five base measures, i.e., *OnlyNumbers*, *OnlyAlphabets*, *Date*, *Length*, and *NumberOfDifferent-Symbols*, presented in Fig. 3 and Fig. 4.

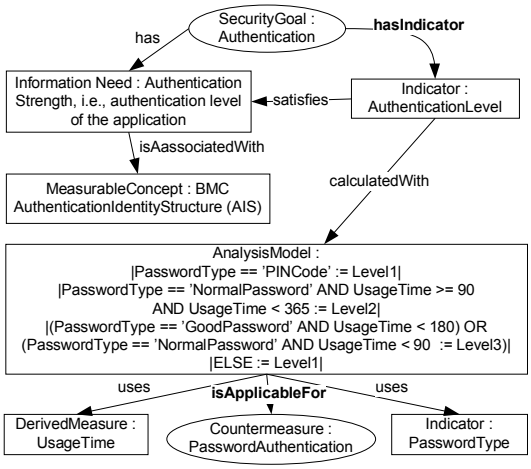


Figure 5. Authentication level.

Currently, the presented analysis models are very simple, utilising only a few base measures, and need to be enhanced in the future. Nevertheless, these analysis models provide the possibility to test the suitability of ISMO for run-time security measurements. Furthermore, the presentation of analysis models in the ontology makes it possible to modify and update them at run-time. The following table lists the mapping properties made from/to instantiated security measures. Again, these mappings are measure dependent. Hence, the addition of a new measure instance also creates new mapping properties.

TABLE II. MAPPING THE PROPERTIES OF INSTANTIATED MEASURES

Concept from OIS [7]	**Mapping property in ISMO (direction)**	**Concept from the SMO** [15]
Threat : Brute-ForceAttack	dependsOn (->)	Attribute : PasswordStructure Attribute : PasswordAge
Credential : Password	hasMeasurable-Attribute (->)	Attribute : PasswordStructure Attribute : PasswordAge Attribute : PasswordLength Attribute : NumberOfDifferentSymbols Attribute : OnlyAlphabets Attribute : OnlyNumbers
Countermeasure : Password-Authentication	relatesToCountermeasure (<-)	Measurable concept : AuthenticationIdentity Structure
Countermeasure : Password-Authentication	isApplicableFor (<-)	Analysis model : analysis model for authentication level

V. THE USAGE OF INFORMATION SECURITY MEASURING ONTOLOGY

This section describes how the ISMO will be used at design and run-time. In addition, ontology evolution is discussed.

A. Utilisation at Design-time

The software designers have to take several issues into account when they design an application that is intended to measure its security level and adapt itself accordingly. Firstly, the required security goals are defined – such as the user authentication. Secondly, the levels for each security goal are defined, e.g., level 1 for security goals which are not very critical and level 5 for extremely critical security goals. It is notable that ISMO does not restrict the number of security levels, for example, the analysis models in the previous section utilised three levels instead of five. Thirdly, the security mechanisms to achieve the required goals are selected, e.g., a username-password pair for authentication. The micro-architecture for run-time security adaptation is presented in [3] – working as a guideline for the software developer by showing the components which are required in an adaptation applicable software.

The OIS already contains mapping properties from goals to supporting mechanisms. However, there is no possibility to define the required levels for the goals. It should be noted that the selection and implementation of security countermeasures is highly context-dependent. The ISMO draws a mapping from the security goal to the level indicator – in our case authentication level – as presented in Fig. 5. Therefore, the software designer can retrieve the base measures from ISMO, used for calculating a particular level indicator. Based on this information, she implements the measuring methods of base measures as the part of application. For example, the authentication level indicator requires five base measures and the related measurement methods as mentioned earlier, which must all be implemented to the application. However, measurement functions and analysis models that combine base measures are not hard coded to the application. Instead, a generic

 ISBN: 978-1-61208-005-5

parser and monitor components are utilised. The parser component retrieves analysis models from ISMO and parses the rules, which depict how the base measures have to be combined. The monitor component utilises these rules and calculates the security level (the authentication level at this time) from the base measures, which is utilised in the software adaptation. The generic and implementation specific parts are presented in Fig. 6. The internal design of these components is not within the scope of this paper.

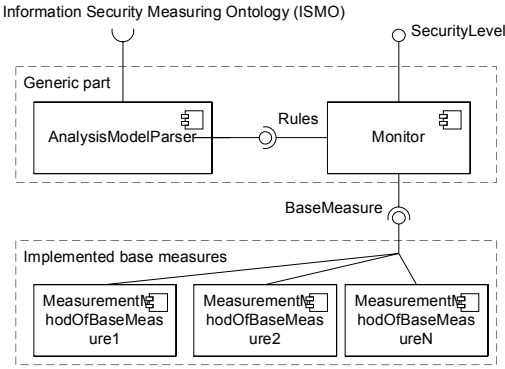


Figure 6. Generic and implementation specific parts.

B. Utilisation at Run-time

The application, which contains a capability to measure its security, is assumed to be aware of its security goal(s) and level(s) and how to measure the fulfilment of its goal(s). Hence, the application retrieves an indicator which is used to measure the level of a particular security goal, e.g., the authentication level indicator for the authentication goal. However, separate analysis models are required for the alternative countermeasures used to achieve user authentication. For example, Fig. 5 contains the analysis model for the password based authentication. Simultaneously, the ontology may contain an analysis model for the security token based authentication, and both of these analysis models are related to the authentication level indicator. Thus, the application must check the currently used countermeasure and select an appropriate analysis model for it from the ISMO. The *isApplicableFor* property maps the analysis model to the countermeasure and makes this selection possible.

Now, the application has a right analysis model. Based on this information, the application searches the measures that are used in the analysis model. The authentication level indicator and the related analysis model, presented in Fig. 5, use the *PasswordType* indicator and the *UsageTime* derived measure. Thus, the application queries ISMO until it finds the base measures which are required to calculate a value for the authentication level indicator. It is notable that these searches only have to be made at application start-ups and when the countermeasure is to be changed at run-time, due to security adaptation demands.

As a result of this search, the application possesses all the information which is required for measuring security. The application has the knowledge of required security goals and levels, and the base measures to be used. Therefore, the application uses measurement methods, which are implemented as a part of it during the design-time. The monitor component (in Fig. 6) combines these base measures to a security level indicator. In a situation where the application is unable to reach the required security level, it adapts the used countermeasure. The results of measuring help to recognise the part of the application that has to be adapted. The security adaptation is discussed in more detail in [21].

It is possible that the required security level changes during the application execution. For instance, the usage of the application may change in a way that requires a higher security level. This type of change does not affect the utilisation of ISMO or the measuring itself. Only the level, compared to the measurement result, changes.

C. Ontology Evolution

At some point, it is necessary to make changes and additions to ISMO. This is required because new threats appear, the usage of the application changes, or the environment of the application changes. The ontology evolution is a challenge from the application point of view, since ISMO is also used for making design decisions. In other words, the required base measures are selected and implemented at the design-time. Thus, a new base measure cannot appear for the application by adding it to ISMO. On the contrary, the analysis models which are used for indicators, such as the authentication level, can be dynamically changed to ISMO. For instance, the analysis model in Fig. 5 defines that level 2 is achieved with a normal password that is used for 3-12 months (90-365 days). However, this can be easily changed to the form: level 2 is achieved with a normal password that is used for 3-6 months. More complicated changes can also be made easily – the only requisite is that the application contains the required base measures. The *AnalysisModelParser* and *Monitor* components (Fig. 6) ensure that changes in the analysis models do not require any changes to the application. However, the analysis models have to be described in a common syntax that the *AnalysisModelParser* is able to parse. ISMO uses simple logical operations to combine the named measurement results, as seen in Fig. 4 and Fig. 5.

VI. A CASE EXAMPLE OF INFORMATION SECURITY MEASURING ONTOLOGY

Run-time security measuring and adaptation was earlier validated in [21, 22] – released on YouTube [23]. Now, a case example is used to exemplify both the design-time and run-time usage of ISMO. The case study takes place in a smart home environment, where the user performs different tasks with her mobile device. The RIBS platform [24] is used to build up the smart home environment. RIBS is a platform that makes it possible for heterogeneous devices to communicate with each other by a means of SIB (Semantic Information Broker) and agents. The SIB is an information

 ISBN: 978-1-61208-005-5

storage where agents publish and subscribe information. The smart home environment contains agents which publish environmental information, for instance, temperatures, humidity, etc. Home automation devices contain agents which subscribe to control information from the home SIB. Furthermore, the smart home environment contains agents, which offer entertainment information for the user, i.e., news, weather forecasts, etc.

In the case study, information from the smart home is utilised with a smart space application, which is running on a Nokia N900 mobile device. The smart space application and the related base measures are implemented using the Python programming language. In this case example, two SIBs exist. The first one (personal SIB) runs on the user's N900, as storage for ISMO. Alternatively, ISMO can be stored in the mass storage of the N900 in an OWL format. The second one (home SIB) runs on a computer in the smart home, and constitutes the home smart space. The application communicates with the personal and home SIB via TCP/IP communication and measures the achieved authentication level by a means of ISMO.

Firstly, ISMO is used at the smart space application design time as described in the previous section. The generic part, i.e., *AnalysisModelParser* and *Monitor* components, are imported to the application. The application developer makes a decision that passwords will be used for authentication and searches the supporting analysis model from ISMO. Furthermore, the base measures which are required in the analysis models are retrieved and implemented to the application. In this case, the used base measures are *OnlyNumbers*, *OnlyAlphabets*, *Length*, *NumberOf-DifferentSymbols*, and *Date*.

Secondly, ISMO is used while running a smart space application. When the user opens the smart space application, the application automatically retrieves ISMO from the personal SIB. The user then opts to join the home smart space with the smart space application. During the join operation, the user is authenticated for the first time and the authentication level monitoring process starts. The *AnalysisModelParser* component reads ISMO. The *Monitor* component receives rules on how to combine different base measures and provides the authentication level for the security adaptation. Both the *AnalysisModelParser* and *Monitor* components are running on the N900. The used countermeasure is password authentication – based on this information, the monitor component selects the correct analysis model to calculate the authentication level. It is notable that the application can contain several authentication mechanisms and ISMO provides information concerning which analysis model to use with each mechanism. The home smart space contains various types of information and the utilisation of different information requires their own authentication levels. Thus, we defined the following authentication requirements for different tasks:

- Level 1 for entertainment usage,
- Level 2 for retrieving information from sensors, etc.,
- Level 3 for controlling building automation devices.

The smart space application is aware of what the user is currently doing, i.e., it monitors the current context and reports this information to the security adaptation. The user decides to login with a username and password on authentication level 2. Thus, the user is unable to control the building automation devices. In an accelerated use case, when the password usage time reaches 12 months, the authentication level decreases to level 1. Hence, the smart space application only provides entertainment information for the user. When the user attempts to perform a task which requires higher authentication level, the smart space application recognises that an adaptation is required. The adaptation asks the user to re-authenticate with a better password, as is shown in Fig. 7. Consequently, the application user does not require any knowledge of ISMO, i.e., the smart space application seamlessly utilises the content of ISMO.

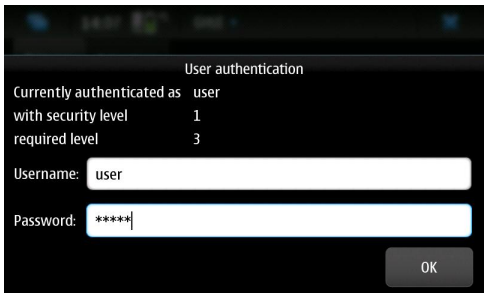


Figure 7. Re-authenticating the user.

The purpose of the case example was to test designed and instantiated ontology. Thus, the content of the analysis models and measures was not within the scope of the case. The utilisation of ISMO ensured that security can be measured in a dynamic environment. Without ISMO, the used analysis models have to be hard coded to the application, which is unreasonable in the dynamic environment. The case example proved that when the *AnalysisModelParser* and *Monitor* components exist, the implementation of the security measures to the application is straightforward. The application developer merely has to implement the required base measures as declared in the ISMO, or use existing base measures. The application developer is able to utilise measures from ISMO, without a need to implement ontology parsers. Moreover, the application was able to retrieve analysis models from ISMO and monitor the authentication level at run-time. The Monitor component calculates a new authentication level each time the used base measures change. However, the *AnalysisModelParser* component checks the content of ISMO at pre-defined intervals.

It is a commonly known issue that ontology searches may cause performance overheads. However, in this case example, the ontology was used in a mobile device without a major overhead. Nevertheless, it is important to optimise how often information is retrieved from ISMO. This helps to achieve the performance requirements of the application,

 ISBN: 978-1-61208-005-5

since changes in ISMO are only checked at pre-defined intervals. Therefore, there is no need to continually query the personal SIB. In the case example, the searches were made every 60 seconds and this kind of checking interval had no visible effects on the usability or performance of the application. Another alternative is to utilise subscriptions, which automatically inform to applications when ISMO is changed. However, the performance overhead of this option is not known beforehand, i.e., changes in ISMO can take place at anytime.

VII. DISCUSSION

In this work, we utilised existing ontologies – instead of starting from scratch – to achieve the information security measuring ontology for run-time usage. Thus, we gained a wide and extensible ontology that is compatible with its predecessors. The combination also ensures a higher maturity level, as the ontologies which were used were already validated. It can be seen from the ontology comparison presented in [10] that the existing security ontologies contain a large deal of overlapping. This work does not add overlapping concepts, which is important from a compatibility viewpoint. Utilisation of the SMO ensures that the measurement part of ISMO is generic. Therefore, the addition of new measures in the future will be easy. Furthermore, the used concepts can also be utilised to measure other quality attributes. Initially, the SMO is not intended for run-time usage. However, there are no constraints to applying the SMO at run-time situations as measuring related terminology is similar in both design and run-time measurements.

It might seem that using ontology to achieve a run-time measuring applicability is a too heavy weight solution. Nevertheless, in cases where an application contains several mechanisms for reaching a particular security goal, it will be necessary to describe the measures in detail. This is particularly necessary when the application is intended to adapt used security mechanisms. In addition, ISMO makes it possible to update and add analysis models – when a new vulnerability is found or the application usage changes. Currently, measurement functions and analysis models are described by using simple logical operations in the ontology – parsed by the *AnalysisModelParser* component. Logical operations were suitable for the measures used in this work. However, in the future, there is a need for additional mathematical operations, required in security measuring. The ontology definition is made at a level that possesses sufficient detail, and thus, it is possible for ISMO to provide the required knowledge for an autonomous measuring process.

Mapping between OIS and SMO is a complex task due to the complexity of measuring security. Currently, a concept level mapping is done, but there were only a few concept-to-concept mappings, which enforces the creation of mapping from/to instantiated security measures. Authentication related measures are instantiated to ISMO as an example. Additional mappings are required when a new measure instance is added. However, the measure instances added in this work offer an example of how to add the mappings, and thus, facilitate future additions. It is notable, that different types of measures will create entirely different mappings between these ontologies. For example, risk measures will create mappings between assets from the OIS and attributes from the SMO. On the other hand, there is not always a mapping property from the attribute concept (in SMO) to some specific credential (in OIS). Hence, mappings between these ontologies depend on the security goal, the used security mechanism, and the used measure.

The performed case example showed that ISMO can be used even in a mobile device without a major performance overhead. However, a more thorough performance evaluation has to be performed in the future. One question is how the usage of ISMO affects the achieved security. For instance, an attacker may cause constant environment changes, which in turn create a set of queries for ISMO and might jeopardize the availability. In addition, measurement methods and results might also be the target of an attack. Thus, it is necessary to perform the run-time measurement in a way that supports the achievement of security requirements, instead of creating new vulnerabilities and possibilities for attacks.

Survey of adaptive application security in [25] lists few adaptation approaches. Added to these, the Extensible Security Adaptation Framework (ESAF) [26] utilises security policies to adapt security mechanisms in the middleware layer. Furthermore, adaptation for Secure Socket Layer (SSL) is presented in [27][28] and monitoring for Java ME platform in [29]. The ISMO offers several advantages compared to existing self-adaptation and policy based approaches. Firstly, security measuring triggers an adaptation task, instead of beforehand defined situation. Secondly, ISMO is a generic solution, i.e., it is not tied to only one security mechanism, platform, or security goal. Finally, ISMO based approaches are dynamic – new analysis models can be added and the existing ones can be modified.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel ontology – called ISMO – for information security measuring, developed particularly for the needs of run-time security measuring. The main purpose was to achieve an ontology that is able to support security measuring at the run-time of an application. The ontology development utilises two existing ontologies: (i) an ontology of information security, describing security related concepts, and (ii) a software measuring ontology, describing general measuring terminology. Firstly, a conceptual mapping between these ontologies was introduced. However, security measuring is a complex task where only a few concept-to-concept relationships can be made. Secondly, the ontology was instantiated by using password related measures. The measures which were used were simple – password structure and password age – however, these measures offered a good starting point to construct ontology which is applicable to run-time security measurements. After the ontology instantiation, we described how to utilise the ISMO in a way that supports run-time measurements. The case example was utilised to exemplify how to use ISMO in a smart home environment. Finally, we

 ISBN: 978-1-61208-005-5

discussed the advantages and shortcomings related to the designed ontology.

In the future, it is important to evaluate the performance cost of using ISMO. In addition, it is important to add new security measures to ISMO, and test how easily these extensions can be made.

ACKNOWLEDGMENT

This work has been carried out in the SOFIA ARTEMIS and GEMOM EU FP7 projects, funded by Tekes, VTT, and the European Commission.

REFERENCES

[1] D. M. Chess, C. C. Palmer, and S. R. White, "Security in an autonomic computing environment," *IBM Systems Journal, 42(1),* pp. 107-118. 2003.

[2] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho, "Knowledge based quality-driven architecture design and evaluation," *Information and Software Technology, 52(6),* pp. 577-601. 2010.

[3] A. Evesti and S. Pantsar-Syväniemi, "Towards micro architecture for security adaptation," *1st International Workshop on Measurability of Security in Software Architectures (MeSSa 2010),* pp. 181-188, 2010.

[4] ISO/IEC 9126-1:2001. *Software Engineering - Product Quality - Part 1: Quality Model.* 2001.

[5] A. Avižienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, pp. 11-33, 2004.

[6] ISO/IEC 15408-1:2009, *Common Criteria for Information Technology Security Evaluation - Part 1: Introduction and General Model.* International Organization of Standardization, 2009.

[7] A. Herzog, N. Shahmehri, and C. Duma. (2009, "An ontology of information security," In *Techniques and Applications for Advanced Information Privacy and Security: Emerging Organizational, Ethical, and Human Issues,* Eds. H. R. Nemati, pp. 278-301, 2009.

[8] J. Zhou, "Knowledge Dichotomy and Semantic Knowledge Management," *Industrial Applications of Semantic Web,* pp. 305-316, 2005.

[9] C. Blanco, J. Lasheras, R. Valencia-García, E. Fernández-Medina, A. Toval, and M. Piattini, "A systematic review and comparison of security ontologies," *3rd International Conference on Availability, Security, and Reliability (ARES 2008),* pp. 813-820, 2008.

[10] A. Evesti, E. Ovaska, and R. Savola, "From security modelling to run-time security monitoring," *European Workshop on Security in Model Driven Architecture (SECMDA),* pp. 33-41, 2009.

[11] G. Denker, L. Kagal, and T. Finin, "Security in the Semantic Web using OWL," *Information Security Technical Report, 10(1),* pp. 51-58. 2005.

[12] A. Kim, J. Luo, and M. Kang, "Security Ontology for annotating resources," *LNCS,* vol. 3761, pp. 1483-1499, 2005.

[13] P. Savolainen, E. Niemelä, and R. Savola, "A taxonomy of information security for service centric systems," *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007),* pp. 5-12, 2007.

[14] B. Tsoumas and D. Gritzalis. "Towards an Ontology-based Security Management," *20th Advanced Information Networking and Applications 2006 (AINA 2006),* pp. 985-992, 2006.

[15] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement," *Information and Software Technology, 48(8),* pp. 631-644, 2006.

[16] N. F. Noy and D. L. McGuinness. "Ontology development 101: A guide to creating your first ontology,", pp. 1-25 2001.

[17] R. Savola, "A Security Metrics Taxonomization Model for Software-Intensive Systems," *Journal of Information Processing Systems, 5(4),* pp. 197-206, 2009.

[18] R. Savola and H. Abie. "Development of measurable security for a distributed messaging system," *International Journal on Advances in Security, 2(4),* pp. 358-380, 2010.

[19] C. Wang and W. A. Wulf, "Towards a Framework for Security Measurement," *Proceedings of the Twentieth National Information Systems Security Conference,* pp. 522-533, 1997.

[20] SPARQL Query Language for RDF, W3C Recommendation, http://www.w3.org/TR/rdf-sparql-query/, 31.1.2011

[21] A. Evesti and E. Ovaska, "Ontology-Based Security Adaptation at Run-Time," *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO),* pp. 204-212, 2010.

[22] A. Evesti, M. Eteläperä, J. Kiljander, J. Kuusijärvi, A. Purhonen, and S. Stenudd, "Semantic Information Interoperability in Smart Spaces," *8th International Conference on Mobile and Ubiquitous Multimedia (MUM'09),* pp. 158-159, 2009.

[23] Semantic Information Interoperability in Smart Spaces, http://www.youtube.com/watch?v=EU9alk9t7dA, 31.1.2011

[24] J. Suomalainen, P. Hyttinen, and P. Tarvainen, "Secure information sharing between heterogeneous embedded devices," *1st International Workshop on Measurability of Security in Software Architectures (MeSSa 2010),* pp. 205-212, 2010.

[25] A. Elkhodary and J. Whittle, "A Survey of Approaches to Adaptive Application Security," *International Workshop on Software Engineering for Adaptive and Self-Managing Systems, 2007 SEAMS '07.,* p. 16, 2007.

[26] A. Klenk, H. Niedermayer, M. Masekowsky, and G. Carle, "An architecture for autonomic security adaptation," *Ann Telecommun, 61(9-10),* pp. 1066-1082. 2006.

[27] C. J. Lamprecht and A. P. A. van Moorsel, "Adaptive SSL: Design, Implementation and Overhead Analysis," *First International Conference on Self-Adaptive and Self-Organizing Systems, 2007. SASO '07.,* pp. 289-294, 2007.

[28] C. J. Lamprecht and A. P. A. van Moorsel, "Runtime Security Adaptation Using Adaptive SSL," *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium,* pp. 305-312, 2008.

[29] G. Costa, F. Martinelli, P. Mori, C. Schaefer, and T. Walter, "Runtime monitoring for next generation Java ME platform," *Comput. Secur., 29(1),* pp. 74-87. 2010.

 ISBN: 978-1-61208-005-5

PUBLICATION VI

Design time reliability predictions for supporting runtime security measuring and adaptation

In: Proceedings of the Third International Conference on Emerging Network Intelligence (EMERGING), Lisbon, Portugal, 20–25 November 2011. Pp. 94–99.

Design Time Reliability Predictions for Supporting Runtime Security Measuring and Adaptation

Antti Evesti, Eila Ovaska
VTT Technical Research Centre of Finland
Oulu, Finland
{antti.evesti, eila.ovaska}@vtt.fi

***Abstract*—The reliability of a quality-critical software component affects the security level that is achieved. There is currently no runtime security management approach that uses design time information. This paper presents an approach to exploiting design time reliability predictions in runtime security management. The Reliability and Availability Prediction (RAP) method is used to predict reliability at software design time. The predicted reliability values are stored in ontology form to support runtime use. The use case example illustrates the presented approach. The presented approach makes it possible to use design time reliability predictions at runtime for security measuring and adaptation. Hence, the reliability of security mechanisms is taken into account when security adaptation is triggered.**

Keywords - information security; quality; evaluation; metric; architecture

I. INTRODUCTION

A variety of quality prediction and testing techniques are used at software design time. The results of these predictions are used to enhance architecture designs, select better component alternatives, and reveal implementation errors. The use of these prediction results ends when satisfactory quality is achieved for a component or system and the product is delivered. However, these prediction results could also be used in runtime situations. This is reasonable, especially in reliability and security management. Reliability is an important factor in achieving a required security level, as can clearly be seen from the security decomposition presented in [1]. Weak reliability of a security-related software component ruins the offered security. Hence, the reliability information of component is valuable for security-related decision-making. This paper therefore presents an approach to bring the design-time reliability prediction results for runtime security measuring and adaptation purposes. To achieve this, ISMO (Information Security Measuring Ontology) [2] is extended in a way that allows prediction results to be stored at design time.

In the literature, different security adaptation approaches exist. The adaptive SSL presented in [3] sets parameters for the SSL session based on the environment information. An Extensible Security Adaptation Framework [4] adds a middleware layer for security mechanisms. The application sets the required security policy and, based on the policy, the middleware layer selects security mechanisms. Context-sensitive Adaptive Authentication [5] uses time and location information to calculate a confidence level for the authentication. In some situations, a low confidence level is sufficient while others require adaptation of the authentication method used. Our earlier work presents an approach that uses ontologies and risk-based measures for security adaptation [6]. These adaptation approaches are intended to work at runtime by observing the system's resources and environment. Based on the observations, different security mechanisms or parameters are set. To our knowledge, none of the existing approaches uses design-time information for adaptation purposes.

Figure 1. presents the broader context of the contribution of this paper. In the first phase, the Reliability and Availability Prediction (RAP) method [7] is used to predict future reliability from software designs. The prediction results are stored in ontology form in order to ensure exploitation at runtime. In this paper we will focus on this first phase. In the second phase, application security is measured at runtime. Reliability predictions are used as input information for security measuring. The third phase is security adaptation, which is triggered by the measuring phase. The adaptation also uses reliability predictions to select the most suitable security mechanism for different situations. After the adaptation, the execution returns to the measuring phase.

Figure 1. Broader scope

The contribution of this paper makes it possible to use design-time reliability predictions for runtime security measuring and adaptation. Hence, a wider information set is available for triggering and making a decision on the adaptation. In other words, information for runtime use can be collected in different phases of the application lifecycle. Thus, the adaptation is not only based on the measurements made just before adaptation but also on knowledge of the whole life cycle of the component.

The paper is organised as follows. After the introductory section, background information is presented. Next, Section 3 is divided into three parts describing the design steps towards applications with security adaptation, design time

 ISBN: 978-1-61208-174-8

reliability predictions, and a way to transform prediction results into the ontology form. Section 4 illustrates the presented approach by means of a case example. A conclusion and future work ideas close the paper.

II. BACKGROUND

Reussner et al. define reliability as the probability of failure-free operation of a software system for a specified period of time in a specified environment [8]. ISO/IEC defines security as follows: The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them [9].

The RAP method evaluates the reliability of a designed software system and its components already at architecture design time [7, 10]. The RAP method reveals design flaws and critical components from the reliability viewpoint. The evaluation is based on architectural models, which means that the first evaluation results are available before any implementation effort is required. Hence, modifications can be performed easily. The RAP method uses state-based models, i.e., Markov models, to predict the reliability of components. The path-based models are used to predict the reliability of a single execution path and the whole software system. The RAP method produces the following reliability values, known as probability of failure (pof) values: 1) independent pof values for software components, 2) pof values for execution paths, 3) the component's pof value in each execution path, 4) the components' system-dependent pof values, and 5) the pof value for the whole software system. The RAP method supports the feedback loop from software testing [11]. The prediction results can therefore be replaced with more accurate values when measured reliability values are available from the software testing. Tool support for the RAP method, called the RAP tool, is also available. The RAP tool reads architectural models from UML diagrams, i.e., state, component, and sequence diagrams. In addition, the RAP tool uses usage profiles that describe system usage, i.e., how many times each execution path is called. The usage profiles make it possible to perform own predictions for different user groups, e.g., professional and normal users. In this work, the results from the RAP tool will be made available for runtime use.

Evesti et al. present the ISMO ontology in [2]. The ISMO composes security ontology and general software measuring terminology. The ISMO thus offers a generic and extendable way to present security measures. These measures are connected to security threats and/or supporting mechanisms, depending on the measure. Measures are divided into base measures, derived measures, and analysis models. The base measure is the simplest measure and is used for more complex measures, i.e., derived measures and analysis models. The ISMO is instantiated as an example using authentication measures, especially Authentication Identity Structure (AIS) measures [1] for password-based authentication. The ISMO thus contains measures for password age and type, i.e., length and the number of different symbols. The software application uses different measures from those of the ISMO to measure its security level at runtime. In this work, the ISMO is extended to contain design time reliability predictions.

Savola et al. present Basic Measurable Components (BMCs) for security attributes (e.g., authentication, confidentiality, etc.) in [1]. BMCs are derived by means of the decomposition approach. The idea of BMCs is to divide security attributes into smaller pieces that can be measured. For example, authentication is divided into five BMCs in [1] as follows: Authentication Identity Uniqueness (AIU), Authentication Identity Structure (AIS), Authentication Identity Integrity (AII), Authentication Mechanism Reliability (AMR), and Authentication Mechanism Integrity (AMI).

III. RELIABILITY PREDICTIONS FOR SUPPORTING SECURITY MEASURING AND ADAPTATION

This section is dived into three subsections. Firstly, high-level design steps for the application with security adaptation features are described. Secondly, a design time reliability prediction is presented. Finally, a way to store the prediction results in ISMO in a way that supports runtime measuring is described.

A. Designing an Application with Security Adaptation Features

This subsection lists design steps that a software architect has to take when designing an application with security adaptation features. Figure 2. illustrates these design phases. The last three phases of the process are iterative. This is not depicted in the figure, however, for reasons of clarity.

1) Required security attributes

In the first phase, the software architect has a set of required security attributes for the application, for instance, S1 for communication confidentiality, S2 for user authentication, and S3 for data integrity requirements. S refers to a security requirement in general.

2) Adaptable security attributes

The software architect has to design adaptation features separately for each security attribute. From the above-listed required security attributes, the architect has to select which ones to implement in an adaptable manner, i.e., variation will take place at runtime [12]. In Figure 2. user authentication S2 is selected for the adaptable security attribute. Other security attributes are thought of as static security requirements from the runtime viewpoint. In other words, the possible variation in these attributes is taken into account at design time.

3) Mechanisms for adaptable security attributes

The adaptable security requirement has to be met by security mechanisms that can be changed or that have parameters that can be modified at runtime. For example, in the adaptable user authentication case, the architect designs two alternative user authentication mechanisms for the application, e.g., password-based and voice-based authentications. Another alternative is to design one security mechanism and set different parameters for it at runtime.

4) Measurements for triggering adaptation

 ISBN: 978-1-61208-174-8

Software measures are designed for the application in parallel with the mechanism design phase. In particular, this means base measures that require measuring probes inside the application. Adaptation at runtime will be triggered based on derived measures and analysis models, which both depend on base measures. In other words, base measures are used to compose derived measures and analysis models. Hence, the software architect has to implement these base measures for the application.

5) Architecture design

The architect designs the architecture for the system. From the security adaptation viewpoint, it is important that variation points are designed with care. For adaptable user authentication, this means that an authentication feature can be called without knowing the currently used authentication mechanism.

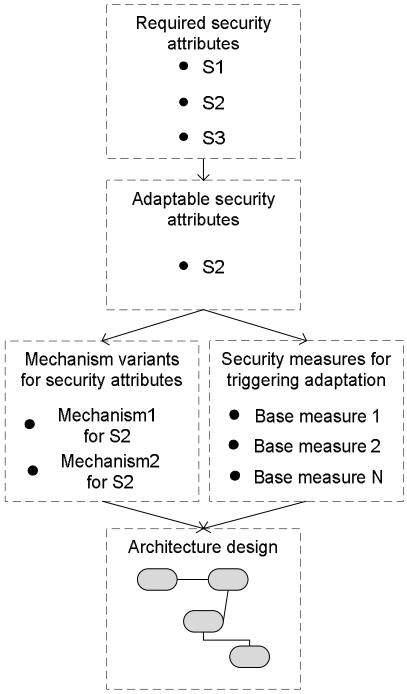


Figure 2. Steps towards adaptable application

B. Design Time Predictions

This subsection describes how the software architect predicts the reliability of components from the architectural designs. The architect uses the RAP method to perform these predictions. Based on the steps listed in the previous subsection, the architect has design documents for the application. Firstly, the component diagram describes the structure of the application. Secondly, the internal behaviour of components is described by means of state diagrams. Finally, sequence diagrams describe the mutual behaviour of components, i.e., how the component calls other components.

The RAP method contains state-based and path-based reliability prediction methods. For runtime security measuring and adaptation purposes, the RAP method is used to predict the probability of failure (pof) values for security mechanism components, i.e., mechanisms designed in phase 3 of the previous subsection.

The state-based prediction method calculates reliability for one independent software component by means of state diagrams. In state diagrams, pof values are given for each state to describe the failure probability in that particular state. Moreover, transition probabilities between states are described. Based on this information, the RAP tool automatically adds a separated failure state and calculates the component's pof value using a state transition matrix *p* and a probability vector *p(n)* as follows:

$$p = \begin{bmatrix} p_{SS} & p_{SA} & p_{SB} & p_{SF} \\ p_{AS} & p_{AA} & p_{AB} & p_{AF} \\ p_{BS} & p_{BA} & p_{BB} & p_{BF} \\ p_{FS} & p_{FA} & p_{FB} & p_{FF} \end{bmatrix} \quad (1)$$

$$p(n+1) = p(n)*p \quad (2)$$

In transition matrix p, p_{SA} presents the probability of transit from the start state S to state A. Similarly, p_{AF} presents a probability of transit from state A to the failure state F. In the beginning, the probability vector takes the form p(0) = [1, 0, 0, 0], which means that the probability of being in the start state is 1 at time moment 0.

The state-based prediction produces independent pof values for the components. These values are further used to calculate the component's pof values in different execution paths. Execution paths are presented by means of sequence diagrams in architectural models. The following equation is used to calculate a component's pof value in a particular execution path:

$$p_{ij} = 1 - (1 - p_i)^{Nij} \quad (3)$$

The previously calculated independent pof of the component is substituted in p_i, and Nij represents the number of execution times of the component in that execution path. Execution paths describe how the particular component is called in different execution paths.

As mentioned in Section 2, the RAP tool is also able to calculate pof values for each execution path, the component belonging to the particular software system, and for the whole software system. The equations for these calculations are presented in [11]. However, our interest is in bringing the previously presented component-related pof values for runtime use.

C. Storing Prediction Results in a Runtime-Applicable Way

After the RAP predictions, the software architect has the components' independent pof values and the components' pof values for the execution paths. Initially, the RAP tool was only intended for use at design time. Thus, the RAP tool

 ISBN: 978-1-61208-174-8

stores these reliability values in the component diagram by means of a UML profile. Hence, the values are available during the implementation and testing phases. Figure 3. shows the security mechanism part of the component diagram after the RAP predictions. Now, the mechanism alternatives designed earlier contain the predicted pof values. This is not practical for runtime purposes however. Reading the pof values from the UML profile requires a connection to a UML tool, which cannot be offered at runtime.

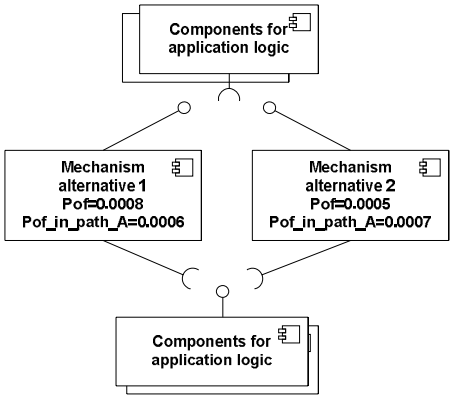


Figure 3. Component diagram after reliability predictions

As mentioned in Section 2, the ISMO supports runtime security measurements. The ISMO is therefore extended to store the components' reliability values. The following information needs to be stored in the ISMO:

1. Software component name
2. Software component version number
3. Which security mechanism the component implements
4. Reference to a place where the pof values are stored
5. Information on the execution path used to calculate path-specific pof values

The component name is intended to separate different alternatives of the mechanisms and is the name taken directly from the component diagram. It is natural to create an instance in the ISMO with a component name. This is because each software component is an individual element.

The version number separates different implementations of the same component. For instance, a new component version that contains bug fixes has a better pof value than the old version. This information therefore has to be separated in the ISMO. The version number is combined with the component name, i.e., an instance name in the ISMO. This naming convention also ensures that the ISMO does not contain instances with the same name.

Information on the security mechanism that the component implements is required because components use different security mechanisms to meet the required security, i.e., the mechanism alternatives in Figure 3. use different security mechanisms. For example, two components can use different authentication mechanisms to achieve user authentication. Countermeasures are described as concepts, i.e., classes, in the ISMO. Thus, it is reasonable to create the instance from the software component under the right countermeasure concept. Figure 4. presents instances created from software components from different countermeasure concepts.

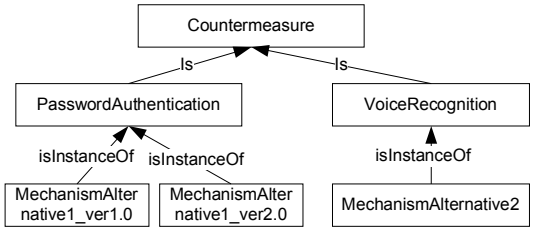


Figure 4. Component instances in the ISMO

The most important information is the components' pof values, and the above-described additions to the ISMO make it possible to put this information into the ISMO. Figure 5. shows a way of presenting pof values in the ISMO. A new base measure called *pof* is added to the ISMO. This is able to offer the component's pof values for the runtime measuring. In the ISMO, each measure is defined for *attribute*, i.e., path-specific pof and independent pof. The attribute relates to *MeasurableConcept*, i.e., Authentication Mechanism Reliability (AMR). Previously, the ISMO contained only measures related to the Authentication Identity Structure (AIS). Both attributes are connected to the countermeasure instance, i.e., MechanismAlternative_ver1.0 in this case, with the *hasMeasurableAttribute* property. Other mechanism instances also contain these attributes. However, for reasons of clarity, these are not presented in the figure.

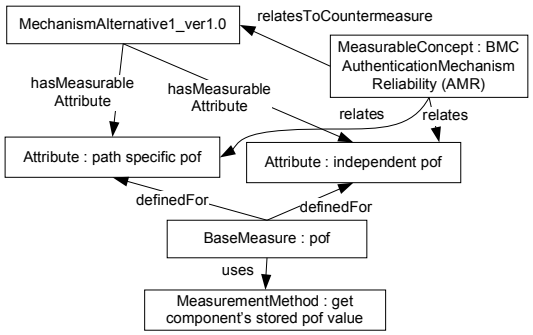


Figure 5. Update for the ISMO

Both attributes use the same pof base measure. The purpose of this base measure is to use *MeasurementMethod*, which retrieves the components' pof values. The measurement method is a concrete measuring probe that is able to retrieve pof values. Hence, it has to know the format

 ISBN: 978-1-61208-174-8

that is used to store pof values. The following structure is used: componentName, componentPof, the component's pof in execution path 1, the component's pof in execution path 2, etc. This structure therefore offers information on the execution path used to calculate path-specific pof values. It is possible to store pof values in a separated file or structure inside the application code. The separated file offers more flexibility, however, i.e., pof values can be updated without knowing the program code. The architect decides where the pof values are stored and creates an appropriate measurement method.

The reason why pof values are not stored directly in the attributes is twofold. Firstly, the measurement part of the ISMO – inherited from the Software Measurement Ontology (SMO) [13] – defines that attributes only define things that can be measured. Secondly, storing pof values outside the ISMO makes the ontology and pof values manageable. An application with security adaptation can contain several security mechanism components and each component can belong to several execution paths. Storing all these values into the ISMO will increase its size and complicate the updating of pof values.

IV. USE CASE EXAMPLE

This section gives a use case example of the presented approach. The purpose of the example is to show how the reliability of the security mechanism component is predicted. The results are stored in a runtime-applicable way in the ISMO.

The software architect designs a software application with security adaptation features. Communication confidentiality and user authentication are required securities for the application, c.f. Figure 2. From these security requirements, it is decided to implement user authentication in an adaptable manner. Hence, the architect designs alternative mechanisms for achieving user authentication, for example, password-based and fingerprint authentication. At the same time, base measures for measuring the user authentication are designed for the application. One of these base measures is pof. The value of the pof base measure is retrieved using a measurement method. It is notable, that the base measures and related measurement method implementations are reusable. Hence, the same base measure is also applicable to other security mechanisms.

After these design steps, there will be a component diagram, state diagrams of components, and sequence diagrams. Both authentication mechanisms are implemented as one independent software component called *passwordAuthentication* and *fingerprintAuthentication*.

Figure 6. presents a state diagram for the password authentication component. In this case, each transition probability is 1, i.e., only one leaving transition from each state. The architect sets the pof values for each state heuristically, and these pof values then affect the transition probabilities. In other words, the state's pof value reduces the occurrence probability of the right state transition respectively. Based on values from Figure 6. the RAP tool automatically adds the failure state and builds the transition matrix p as described in Section 3. From the transition matrix, the RAP tool calculates the pof value for the *passwordAuthentication* component. In this case, the pof value for the *passwordAuthentication* component is 0.000482. Similarly, pof values are given for states in the *fingerpringAuthentication* component, and the pof value of the component is calculated.

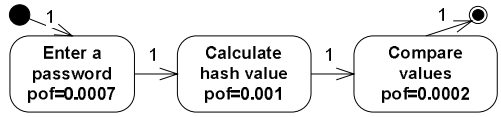


Figure 6. State diagram for the passwordAuthentication component

To exemplify path-specific pof values, the sequence diagram in Figure 7. is used. The RAP tool uses this sequence diagram, previously calculated pof value, and equation 3 to calculate the path-specific pof value. Hence, a pof value of 0.000482 is attained for the *passwordAuthentication* component in this specific execution path. In this case, the independent and path-specific pof values are the same because the *passwordAuthentication* component is only called once in this sequence diagram, c.f. equation 3.

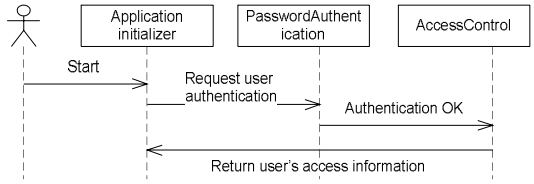


Figure 7. Sample execution path for password authentication

The architect stores this information in the ISMO in the form defined in the previous section and illustrated in Figure 8. In the figure, grey is used to describe information added in this case example. The component name is now *passwordAuthentication* and the version number is 1.0. Hence, the instance named *passwordAuthentication_ver1.0* is created under the password authentication concept in the ISMO. Similarly, the instance for the *fingerprintAuthentication* component is created. Both of these instances contain previously mentioned attributes. Attributes for the *fingerprintAuthentication* are not presented in the figure, however, for reasons of clarity. Calculated pof values are stored in the specific file called pofs. This file is presented in dark grey in Figure 8. because it is a separate part from the ISMO. *MeasurementMethod* contains a link to that file and is able to read pof values from the file. In this case, the file contains pof values for the *passwordAuthentication* and *fingerpringAuthentication* components.

 ISBN: 978-1-61208-174-8

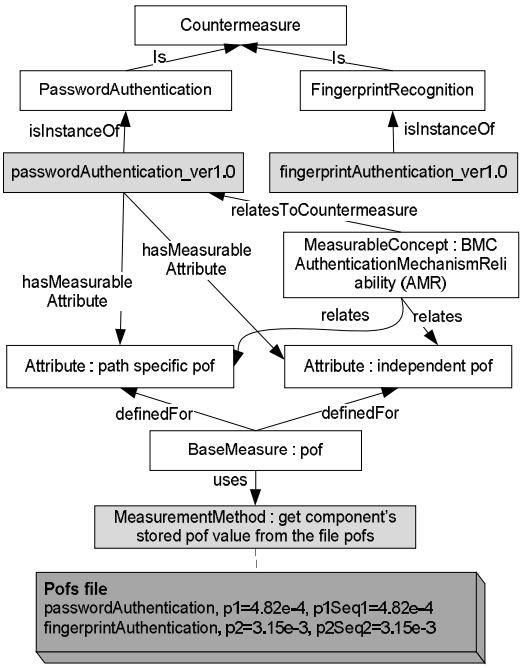


Figure 8. The content of ISMO after design time predictions

V. CONCLUSION AND FUTURE WORK

There is a clear connection between software reliability and security. An unreliable software component that performs security-related actions can ruin the security of the whole application. In this work, an approach was introduced to bring the results from design-time reliability predictions for runtime security measuring and adaptation purposes. Hence, the reliability of the security mechanisms can be taken into account when security adaptation is triggered. The work presented steps on how to produce an application with security adaptation features. Thereafter, reliability was predicted from design documents. Finally, these prediction results were stored in the ISMO, which makes it possible to use the prediction results at runtime. Storing the components' pof values in the ISMO required some extensions to the ontology. Firstly, the way to present individual security mechanism components in the ISMO was added. Secondly, the attributes for pof values were added and finally, a new base measure for pof values was introduced in the ISMO.

To our knowledge, there is no security measuring and adaptation approach that also uses design time information. Thus, the introduced approach is the first step towards enabling the use of the design time reliability predictions for runtime security measuring and adaptation. Reliability values are stored in a way that supports fast and easy updating. This is important when bug fixes for the security components are made. Furthermore, the real use of a component may produce different reliability to that initially predicted and it is then important to update the pof values. The presented approach is not restricted to one particular security mechanism or attribute. Hence, the software architect can make the decision of which attributes will be implemented in an adaptable manner on a case-by-case basis.

In the future, it is important to develop security measures that use the components' pof values in runtime security measuring. Current pof values of components can be used to compare different security components. Moreover, combining the reliability information and security level supported by the component offers valuable information for adaptation purposes. This means that the ISMO will be enhanced by new analysis models. The RAP tool also needs new features for storing information automatically to the ISMO.

ACKNOWLEDGMENT

This work is being carried out in the ARTEMIS SOFIA project funded by Tekes, VTT, and the European Commission.

REFERENCES

[1] R. Savola and H. Abie. "Development of measurable security for a distributed messaging system", *International Journal on Advances in Security, 2(4),* pp. 358-380, 2010.

[2] A. Evesti, R. Savola, E. Ovaska, and J. Kuusijärvi, "The Design, Instantiation, and Usage of Information Security Measuring Ontology", *MOPAS'2011,* pp. 1-9, 17th April, 2011. 2011.

[3] C. J. Lamprecht and A. P. A. van Moorsel, "Runtime Security Adaptation Using Adaptive SSL", *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium,* pp. 305-312, 2008.

[4] A. Klenk, H. Niedermayer, M. Masekowsky, and G. Carle, "An architecture for autonomic security adaptation", *Ann Telecommun, 61(9-10),* pp. 1066-1082. 2006.

[5] R. Hulsebosch, M. Bargh, G. Lenzini, P. Ebben, and S. Iacob. "Context sensitive adaptive authentication", *Smart Sensing and Context,* pp. 93-109, 2007.

[6] A. Evesti and E. Ovaska, "Ontology-Based Security Adaptation at Run-Time", *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO),* pp. 204-212, 2010.

[7] A. Immonen, "A method for predicting reliability and availability at the architecture level", in *Software Product Lines* T. Käkölä and J. Dueñas, Eds., 2006.

[8] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures", *J. Syst. Software, 66(3),* pp. 241-252. 2003.

[9] ISO/IEC 9126-1:2001. *Software Engineering – Product Quality – Part 1: Quality Model.* 2001.

[10] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho, "Knowledge based quality-driven architecture design and evaluation", *Information and Software Technology, 52(6),* pp. 577-601. 2010.

[11] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software", *J. Syst. Software, 84(6),* pp. 1054-1070. 2011.

[12] E. Niemela, A. Evesti, and P. Savolainen, "Modeling quality attribute variability", *ENASE – Proc. Int. Conf. Eval. Novel Approaches Software Eng.,* pp. 169-176, 2008.

[13] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement", *Information and Software Technology, 48(8),* pp. 631-644. 2006.

 ISBN: 978-1-61208-174-8

PUBLICATION VII

Architecture and knowledge-driven self-adaptive security in smart space

In: Computers, Vol. 2, No. 1, pp. 34–66.

Computers **2013**, *2*, 34-66; doi:10.3390/computers2010034

OPEN ACCESS

computers

ISSN 2073-431X

www.mdpi.com/journal/computers

Article

Architecture and Knowledge-Driven Self-Adaptive Security in Smart Space

Antti Evesti [1,*], Jani Suomalainen [2] and Eila Ovaska [1]

1 VTT Technical Research Centre of Finland, Kaitoväylä 1, 90571 Oulu, Finland;
E-Mail: eila.ovaska@vtt.fi (E.O.)

2 VTT Technical Research Centre of Finland, Vuorimiehentie 3, 02044 Espoo, Finland;
E-Mail: jani.suomalainen@vtt.fi (J.S.)

* Author to whom correspondence should be addressed; E-Mail: antti.evesti@vtt.fi;
Tel.: +358-20-722-2101; Fax: +358-20-722-2320.

Received: 26 November 2012; in revised form: 24 February 2013 / Accepted: 4 March 2013 / Published: 18 March 2013

Abstract: Dynamic and heterogeneous smart spaces cause challenges for security because it is impossible to anticipate all the possible changes at design-time. Self-adaptive security is an applicable solution for this challenge. This paper presents an architectural approach for security adaptation in smart spaces. The approach combines an adaptation loop, Information Security Measuring Ontology (ISMO) and a smart space security-control model. The adaptation loop includes phases to monitor, analyze, plan and execute changes in the smart space. The ISMO offers input knowledge for the adaptation loop and the security-control model enforces dynamic access control policies. The approach is novel because it defines the whole adaptation loop and knowledge required in each phase of the adaptation. The contributions are validated as a part of the smart space pilot implementation. The approach offers reusable and extensible means to achieve adaptive security in smart spaces and up-to-date access control for devices that appear in the space. Hence, the approach supports the work of smart space application developers.

Keywords: architecture; authentication; authorization; ontology; self-adaptation

1. Introduction

The smart space trend has initiated several research projects and publications, from technologies to applications. Smart spaces—such as smart homes, smart buildings and smart cities—offer available information and devices for end-users' purposes without a pre-defined configuration or application behavior. Smart spaces are dynamic, and moreover, utilized technologies are heterogeneous. Consequently, it is not possible to envision all the possible situations where the smart space application will be utilized, which creates a challenge for software designers. Self-adaptation is a possibility to respond to this challenge by postponing decision making from design-time to runtime. Self-adaptation is a software's capability to configure and tune its functionality at runtime, in order to tackle changing situations. Challenges related to smart space security are complex—due to the dynamicity and heterogeneity of smart spaces. Firstly, the required security objectives vary between situations, e.g., sometimes integrity is the essential security objective but in other situations the user's privacy is the first priority. Secondly, the security needs of a smart space application vary between situations. For instance, an application utilizing entertainment or critical control information has variable requirements for security effectiveness, *i.e.*, for the security level. Thirdly, smart spaces change continuously, as new devices appear and leave. These devices must be able to use the smart space's available security mechanisms. However, the same security mechanisms are not applicable in all smart spaces. For example, one smart space may support only one particular authentication mechanism while another provides three different mechanisms.

These challenges demand an adaptive security solution that is able to change and modify the used security mechanisms autonomously at runtime. The importance of self-adaptation in smart spaces is also recognized in [1]. A reference model called MAPE-K (Monitor, Analyze, Plan and Execute) has been introduced as a solution for self-adaptation. This model is generic, *i.e.*, it can be applied for various quality and functionality adaptations, and thus it was selected as the reference model for security adaptation in this work. In the MAPE-K model, the Monitor collects information that is analyzed to recognize adaptation needs. Thereafter, the Plan phase creates an adaptation plan for execution. These four phases constitute an adaptation loop supported by knowledge. Currently, several security-adaptation approaches exist [2,3]. The first survey reveals that the existing security-adaptation approaches concentrate on specific security objectives. The second survey shows that the existing approaches have a lack in the adaptation loop coverage, *i.e.*, the approaches do not define the whole MAPE loop. Moreover, Yuan *et al.* note that the abstract architecture for security adaptation is not presented in the existing approaches [3]. These architecture-level problems complicate the reusability and extensibility of the existing security-adaptation approaches. The MAPE-K model separates knowledge from the adaptation loop. However, existing security-adaptation approaches do not support this separation but utilize hard-coded adaptation decisions, which is not sufficient in dynamic smart spaces. Moreover, smart spaces require flexible access control, which is able to handle the situations when new devices and information constantly appear in the smart space. However, the existing approaches are not able to offer this flexibility for dynamically changing access control needs.

This paper concentrates on architecture, knowledge and access control problems. Hence, the paper makes the following contributions: (1) Architecture for the security-adaptation loop is defined and mapped to the MAPE model. Hence, the architecture covers all the adaptation phases in a clearly

defined form. The adaptation loop selects from the security mechanisms and configures the parameters of those mechanisms at runtime. (2) The knowledge is mapped to the security-adaptation architecture, in order to encompass the knowledge part of the MAPE-K reference model. In our solution, knowledge is made accessible to the security adaptation by means of ontologies. Compared with the existing approaches, our solution is novel because it makes it possible to minimize the amount of hard-coded knowledge and supports knowledge addition and modifications. (3) A runtime security-control model for dynamic authorization and access control is defined. The security-control model defines how shared information is effectively secured and controlled in smart spaces.

The novelty of contributions comes from the integration, which builds up the consistent security-adaptation architecture. The architecture covers all adaptation phases from monitoring to execution with dynamic access control. In contrast to the existing approaches, knowledge is separated from the architecture, and mappings from the architecture to knowledge are presented. Thus, the presented security adaptation does not require hard-coded rules for the analysis and planning phases because ontologies form the knowledge for the adaptation. Finally, the whole approach is validated as part of the wider smart space pilot implementation.

Background information and related work are described in Section 2. The whole adaptation approach is presented in Section 3. Section 4 describes the implementation of the approach by means of a use case. A discussion of the approach and future research are summarized in Section 5. Finally, the Conclusions Section closes the paper.

2. Background and Related Work

2.1. Smart Spaces

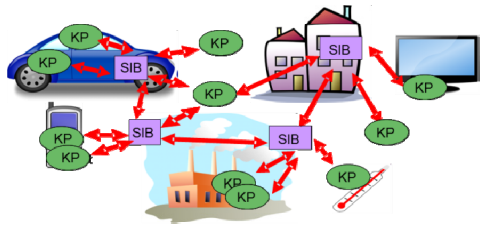
Smart spaces are physical spaces where devices cooperate and share information to intelligently provide services for the users. Cook *et al.* define a smart environment as one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment [4]. The terms smart space and smart environment are widely used interchangeably—this article uses the term smart space.

Cooperation and information sharing requires that devices and Smart Space Applications (SSAs) are able to interoperate. This interoperation is able to occur at different levels, which are called interoperability levels, defined in [5,6]. The presented interoperability levels from bottom to top are Connection, Communication, Semantic, Dynamic, Behavioral and Conceptual interoperability. The Connection interoperability level focuses on network connectivity, whereas the Communication level focuses on data syntax. However, these two lower-most interoperability levels are out of the scope of this paper. The Semantic interoperability level concentrates on understanding data from the communication level. Next, the Dynamic level focuses on context changes and the Behavioral level matches actions together. Finally, the Conceptual interoperability level focuses on abstracting; representing easy-to-use knowledge to the other interoperability levels, and making deductions based on data, context and actions. Thus, conceptual interoperability creates meaning from the information, context and behavior in the smart spaces. Hence, this is the level where the "smartness" is built for the smart space.

The SSA consists of a set of software agents that communicate and share information with each other. Therefore, the deployment of the SSA can be distributed to several smart space devices, *i.e.*, agents of the SSA are executed in different devices instead of one centralized device. These agents and the composed SSA act in dynamically changing smart spaces, which may offer a huge amount of information. Context-awareness is a means to handle this information flow in order to provide reasonable information and services for the user. Similarly, security in smart spaces requires context-awareness in order to provide reasonable security for different situations and actions. From the interoperability-level viewpoint, SSAs and context-awareness occur at the Dynamic and Behavioral levels.

Establishing a smart space requires an appropriate infrastructure—in this paper, the Smart-M3 concept [7] will be utilized. In the Smart-M3, the Semantic Information Broker (SIB) forms a backbone for the smart space. The SIB takes care of information sharing between agents—called Knowledge Processors (KPs). Agents are able to make queries and subscriptions, and insert semantic information in the SIB. Consequently, various devices are able to interoperate, *i.e.*, share semantic information, by means of the SIB and agents inside devices, as illustrated in Figure 1. The SIB utilizes semantic web technologies—especially Resource Description Framework (RDF) [8] and SPARQL query language. From the interoperability-level point of view, the SIB embodies the Semantic Interoperability level. In the Sofia project [9], three different Smart-M3 concept implementations were made for different usages. This paper utilizes the implementation called RIBS [10], which contains mechanisms to secure communication and is able to work in resource-restricted devices—such as a WLAN access point.

Figure 1. Smart spaces formed by Semantic Information Brokers (SIBs) and Knowledge Processors (KPs).



Security challenges caused by the dynamicity and heterogeneity of smart spaces are mentioned above. Moreover, traditional security challenges, e.g., key exchange and resource restrictions, are present in the smart spaces. Similarly, openness and free utilization, which are characteristics of smart spaces, affect security. Nevertheless, this paper focuses on security challenges due to dynamicity and heterogeneity by presenting a security-adaptation approach with a dynamic access control.

2.2. Security Adaptation

Kephart *et al.* define autonomic computing as computing systems that can manage themselves by using high-level objectives given by administrators [11]. The autonomic element contains the MAPE-K control loop composed of the Monitor, Analyze, Plan and Execute phases, supported by

Knowledge. A similar structure is also applied in [12,13] as a reference model for autonomic computing. However, the names of the phases vary. For instance, Dobson *et al.* use the terms Collect, Analyze, Decide and Act [12]. In contrast, Psaier *et al.* define a control loop that joins the Analyze and Plan phases into one Diagnosing phase [14]. The loop concentrates on self-healing, which is a form of autonomic computing. However, in this paper the MAPE-K loop will be utilized as a reference architectural model. The terms autonomic computing, self-management and self-adaptive are utilized interchangeably for instance in [13,15]. This article utilizes the term self-adaptive and its short-form adaptive to refer to software's ability to adapt itself at runtime. Adaptability has been defined as the ability of software to adapt its functionality according to the environment and user [16]. From the security viewpoint, functionality means security mechanisms intended to support the required security objectives.

Elkhodary *et al.* survey four approaches to adaptive security in [2]—namely Extensive Security Infrastructure [17], Strada Security API [18], Willow Architecture [19] and the Adaptive Trust Negotiation Framework (ATNAC) [20]. Moreover, the recent survey from Yuan and Malek [3] compares over 30 self-protection approaches. The extensible Security Adaptation Approach (ESAF) distinguishes security mechanisms from the application to the middleware layer [21]. Hence, the application communicates the required security to the middleware without any knowledge of the used security mechanism. In contrast, Context-sensitive adaptive authentication utilizes context information to replace static authentication mechanisms [22]. Hence, in situations where a lower authentication level is sufficient it is possible to utilize other attributes, e.g., location-based authentication, instead of passwords. The GEMOM (Genetic Messaging-Oriented Secure Middleware)—covered also in [3]—offers self-healing and adaptation features to ensure optimal security strength and uninterrupted operation in changing environments and threats [23]. The GEMOM applies the Monitoring part of the MAPE-K model by means of security measuring [24].

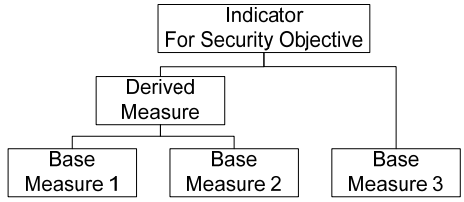
In the security adaptation, the Monitoring phase utilizes security measuring or observes events that affect security. Security measures can be produced by means of the decomposition approach, where security objectives are divided into smaller parts until the measurable components are found [25]. Garcia *et al.* presented a similar technique for generic software measurement in [26] by categorizing measures as Base Measures, Derived Measures and Indicators. Base Measures represent raw measures, which are further composed into Derived Measures. Indicators are on the highest abstraction level—and are able to compose Base Measures, Derived Measures and other Indicators. Figure 2 illustrates the structure of these measures. The indicator for the particular security objective is on the highest level. The Indicator is derived from the Derived measures, Base Measures and other indicators. Structuring measures hierarchically ensures that measures can be reused and extended. Hence, security measures will be applied to Monitoring in this article. The Monitoring phase uses base measures. The results of the base measures are composed in the Analysis phase to reveal the current security level.

The Analyze phase utilizes monitoring results. However, it is not enough to know the current security level but the required security level also has to be known. The required security level can be defined at design-time, or alternatively, the Analyze phase can reason the required level at runtime. Deciding on an appropriate security requirement set is a challenging task. Defining security requirements for design-time purposes is extensively covered in [27]. The presented security

requirements engineering framework takes into account assets, threats, business goals, and system context. All these aspects can be modeled in order to achieve the most extensive security adaptation approach. For example, Salehie *et al.* [28] concentrate on the variability of assets and how this affects security, and how adaptive security is able to deal with these challenges. This is an important viewpoint and starts from assets, which initially set the requirements for all security. It is said that an asset is an entity that someone places value upon [29], and thus, it needs protection. However, we do not model requirements and assets in this granularity. On the contrary, we will utilize context information to recognize the importance and role of handled data, *i.e.*, asset value, which in turn leads to required securities.

Many uncertainties relate to security, and thus, getting accurate numbers to describe security for adaptation purposes is challenging. Sahinoglu [30] utilizes variance values to cover uncertainties in risk analysis. Similarly, in security adaptation, results from monitoring and analysis contain variance in some range. Uncertainties affect the achieved security and the recognized adaptation need but uncertainties do not affect security adaptation architecture or the utilization of knowledge itself. Thus, these uncertainties are out of the scope of this article.

Figure 2. The Structure of Measures.



2.3. Adaptation Knowledge from Ontologies

The MAPE-K model does not define how the knowledge has to be offered. However, in order to follow the separation of the concerns principle, we introduce knowledge as a separately identifiable architectural element by utilizing ontology orientation to represent a self-sufficient model of security concepts. Ontology can be defined as a shared knowledge standard or knowledge model, defining primitive concepts, relations, rules and their instances, which comprise topic knowledge. It can be used for capturing, structuring and enlarging explicit and tacit topic knowledge across people, organizations and computer and software systems [31]. Several security ontologies have been listed in [32]. In addition, our earlier work [33] compared security ontologies from the runtime applicability and measuring viewpoints. Ontologies, designed for runtime usage, often concentrate on the service discovery and matchmaking, e.g., ontologies in [34,35]. However, these ontologies do not cover security measuring. In contrast, Savolainen *et al.* [36] present a security taxonomy for design time usage, which also contains a high-level security measuring part. At the moment, the most extensive security ontology is proposed by Herzog *et al.* [37], known as Ontology of Information Security (OIS). The OIS contains over 250 concepts, which describe security threats, countermeasures, assets and security goals, *etc*. In this paper, security goals and countermeasures are called security objectives and mechanisms, respectively. The OIS lists the following security objectives: confidentiality, integrity, availability, authentication, accountability, anonymity, authenticity, authorization, correctness,

identification, non-repudiation, policy compliance, secrecy and trust. Nevertheless, the OIS does not contain a security-measuring part. In contrast, Garcia *et al.* presented the measurement terminology in an ontology form called Software Measurement Ontology (SMO) in [26]. Consequently, we have combined the Information Security Measuring Ontology (ISMO) from OIS and SMO in [38]. The ISMO makes it possible to present security measures via a common vocabulary and map defined measures to security concepts, e.g., security objectives and mechanisms. In the ISMO, security measures are defined in detail—containing descriptions on how the particular measuring has to be performed and how the base measures can be further combined into indicators. Hence, the ISMO offers knowledge for design-time and runtime purposes, e.g., what kind of measuring probe to implement at design time and how to utilize measuring results at runtime. This paper utilizes knowledge from the ISMO. Furthermore, context knowledge is vital for security adaptation, in order to describe an environment and user actions. For this purpose, we utilize the Context Ontology for Smart Spaces (CO4SS) [39] in this paper. In [40] we defined the taxonomy of context information for security. The taxonomy maps security-related context information to physical, digital and situation context levels. The physical context describes an infrastructure where the SSA is running. The digital context presents the role of the smart space, e.g., public space. Finally, the situation context describes the user's role and activity within the smart space. Moreover, the role of the exchanged/stored data is described in the situation context.

Our earlier work in [41] presented ontology-based security adaptation. In that work, risk-based security measures were stored in the ontology to support security monitoring. Moreover, the ontology contained knowledge about how much each security mechanism decreases the particular security risk, which supports the Planning phase. In [40] we presented a micro-architecture for security adaptation. However, in that architecture the ontology usage was tightly coupled inside the architecture. In this article, the architecture is developed towards the MAPE-K reference model and the ontologies will be separated out to their own interoperability level, *i.e.*, to the Conceptual level.

2.4. Access Control over Semantic Information

To control access to shared semantic information, fine-grained authorization models have been introduced for RDF. These approaches include approaches where access control is implemented as an additional layer on top of the repository, as in [42], and approaches where access-control information has also been integrated into repositories. In the triple-level access control [43], RDF resources are protected with access-restriction properties. Essentially, these properties are links to access policy graphs that specify the owner of the RDF resource as well as those predicates to which this protection applies.

Some researchers have proposed models where RDF-level access control decisions are implicitly derived from existing higher-level policies and context information. The policy-based access control model [44] uses metadata to define permit or prohibit conditions. Jain and Farkas [45] introduce an access-control model where RDF class hierarchy is utilized to manage and derive access control policies. Flourish *et al.* [46] propose a high-level policy-specification language for annotation RDF triples with access-control information. Moreover, approaches for access-control reasoning, based on

concepts and their relations represented by ontologies, have been introduced by Kim *et al.* [47] and Cho *et al.* [48].

However, none of these reasoning solutions are directly applicable for smart spaces. In smart spaces, access control is enforced by information-brokering devices, which are not aware of application-specific policies. Also, semantic reasoning for real-time security control is a challenging task as the reasoning problems are, at the worst case, only solvable in exponential time with respect to the input size [49,50]. Consequently, to enable real-time security enforcement, efficient and scalable solutions are needed. These solutions should enable smart spaces to support different reasoning applications, which may be based on expressive and complex security ontologies.

Our earlier work defined solutions [10,51] for securing communication between smart space devices and controlling information sharing. In [51], we proposed an RDF node-level access-control model for a semantic information broker. The model is simpler than other RDF access-control models as each security policy can be expressed using a single RDF triplet and, in an optimized implementation, be presented with a single bit. In this article, the model is formally defined and generalized, and its granularity is enlarged to protect semantic relationships in addition to semantic information.

2.5. From Quality Variability to Quality Adaptation

In our approach, building a capability for quality adaptation begins at design-time. Consequently, our earlier work combines quality-, model-, and knowledge-driven software development. Our previous work presented the quality-variability model [52]. The variability model defines binding times for variations, *i.e.*, design, assembly, start-up and runtime, which defines the latest time point when quality can be changed. This work concentrates on the situation where security variation occurs at runtime—called security adaptation. Quality variability is closely related to architectures, and thus, the approach for the knowledge-based quality-driven architecture design and evaluation was presented in [53]. The approach contains three steps: (1) Modeling the quality requirement. (2) Modeling the software architecture and transforming the requirements to the models. (3) Quality evaluation. Steps one and two are divided into the knowledge and software engineering processes, whereas step three is divided into the quantitative and qualitative evaluation processes. In that study, quality ontologies for reliability and security were utilized as a knowledge base. Now, we will also bring ontology-based knowledge from design-time for automatic runtime usage.

Lastly, the design steps to produce an application with security adaptation features were presented [54]. The following steps were recognized: (1) Required security objectives—defines all security objectives for the application. (2) Adaptive security objectives—selects objectives, which will be adapted at runtime. (3) Mechanism variants for the selected objectives—selecting security mechanisms and their parameters, which can be adapted at runtime. (4) Measurements for triggering adaptation—selecting security measurements to monitor the adaptation needs of the selected security objectives. (5) Architecture design—designing the selected security mechanisms and measurements into the application architecture in a way that supports runtime adaptation. The contribution of this paper relates to step five, *i.e.*, presenting security-adaptation architecture and mapping it to the knowledge retrieved from the ontologies.

3. The Concept for Adaptive Security

The security-adaptation approach is presented in this section. Firstly, we give an overview of the approach, and thereafter, each part of the approach is presented in its own sub-section. The approach is not bound to any particular security objective or security mechanism. Nevertheless, the approach contains all the necessary components required to build adaptive security for smart spaces. The presented concept is instantiated by means of a case study in Section 4. The case study illustrates the approach from the authentication and authorization viewpoints.

The adaptation approach combines solutions from different interoperability levels. Figure 3 illustrates the proposed solutions—mapped to the interoperability levels and design-time and runtime phases. Sub-Sections 3.1—3.3 concentrate on these levels one-by-one.

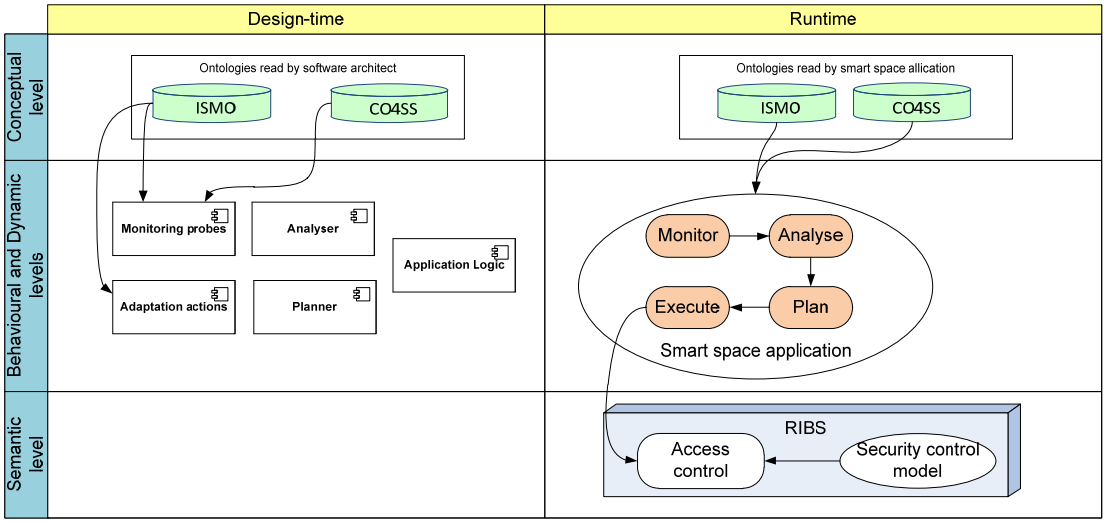
Knowledge, required in security adaptation, is set on the Conceptual interoperability level. The knowledge is described with ontologies—namely the Information Security Measuring Ontology (ISMO) and the Context Ontology for Smart Spaces (CO4SS). Hence, security- and context-related knowledge is offered to other interoperability levels from the Conceptual level at runtime. Ontology-based knowledge ensures that smart space applications (SSA) are able to understand security and context situations in a uniform way. Moreover, the architect utilizes knowledge from ontologies at design-time to implement the appropriate monitoring probes and adaptation actions for the SSA, *c.f.*, connections from ontologies to Monitoring probes and Adaptation actions in Figure 3. In other words, these components are specific for the designed SSA. In contrast, the Analyzer and Planner components search required knowledge at runtime.

The security-adaptation loop—based on the MAPE reference model—and the SSA are located on the Behavioral and Dynamic interoperability levels. The architect designs and implements the required software components at design-time. These components are as follows: Pure application logic, monitoring probes, security mechanisms with adaptation actions, analyzer and planner. The adaptation logic is located in the Planner component. However, the architecture does not dictate the utilized adaptation logic, *i.e.*, the internal functionality of the Planner component. The plan phase is described in Subsection 3.2.3. At runtime, the SSA utilizes monitoring probes to observe security and context changes. The Analyzer component analyses the consequences of the changes based on knowledge retrieved from the ontologies at runtime. Sequentially, the Plan component creates an adaptation plan for responding to changes. Finally, the adaptation plan is enforced by the executors. In the security adaptation these executors are the selected security mechanisms. For example, "a new mobile device has arrived in the smart space" is a change observed by means of monitoring. The SSA analyzes the security consequences of a new device and decides how to adapt to this change. Adaptation actions can, e.g., affect information sharing on the semantic level or reset the communication parameters.

On the Semantic interoperability level, the RIBS constitutes the smart space infrastructure. The RIBS takes care of information sharing between smart space devices and SSAs. Hence, various devices are able to interoperate via the RIBS by sharing semantic information, which is presented by means of RDF. The Semantic level has to contain security solutions, which protect and control the sharing of semantic information. Therefore, we propose a semantic level security-control model to enable efficient access control. The security-control model enables SSAs to prepare security policies so that the RIBS does not have to support complex ontologies or perform runtime security reasoning.

Applications requesting and providing information must unambiguously define the semantics of shared information. However, the presented dynamic access control ensures that variations are handled at runtime, without design-time rules.

Figure 3. Interoperability levels and proposed solutions.



At this point, it is necessary to emphasize the difference between semantic and conceptual interoperability levels. However the semantic technologies, *i.e.*, RDF and OWL, are applied on both levels the difference comes from the abstraction level of the knowledge. The Semantic level contains separated pieces of information, e.g. temperature is minus five or a password length is seven characters. In contrast, the Conceptual level makes it possible to deduct the causes of the semantic information, e.g., water will freeze or the authentication level is low.

3.1. Security Adaptation Concepts from Ontologies

The right knowledge is an essential part of the security adaptation. In our solution, knowledge will be offered from the Conceptual interoperability level by means of ontologies. Ontologies make it possible to update and extend the existing knowledge. Moreover, ontologies support reusability and offer knowledge in a machine-readable form. Knowledge requirements for the security adaptation are threefold, *c.f.*, Figure 4. Firstly, security knowledge is needed to describe security mechanisms, security objectives, threats and their relationships. Secondly, measuring concepts are needed to monitor the achieved security, *i.e.*, the current and/or past security. Thirdly, context knowledge is needed to describe the state of the smart space from situational, digital and physical viewpoints. The situational context is intended to describe the user's role in the smart space and the role of the exchanged/stored data. In contrast, the digital context depicts the role of the smart space. Lastly, the physical context presents the execution platform and smart space infrastructure. The knowledge is arranged in two separate ontologies, *i.e.*, ISMO [38] and CO4SS [39], which together contain over

300 concepts and their connections. The ISMO describes security and measuring knowledge, while CO4SS contains context knowledge.

Figure 4 summarizes the main dependencies of these ontologies, and thus, it contains only the main concepts. The additional concepts and connections are presented in Figures 6–10 in Subsection 3.2. It is notable that the content in Figure 4 is laid in the conceptual level in Figure 3 (the highest level). The context knowledge from the CO4SS sets the required security objectives and levels. For example, dealing with professional information in an office environment has different security requirements than handling the same information in a public environment, which may contain additional threats. The ISMO describes which security mechanism supports the particular security objective and how the security objectives mutually relate. However, the applicable mechanisms depend on the physical context of the smart space, which describes the execution platform and operating system. For instance, the used operating system supports only a particular security mechanism, therefore, in Figure 4 the Physical context is connected to Security mechanisms with an offers connector.

Triggering the security adaptation requires that both security and context concepts are monitored. The monitoring focuses on measurable attributes (attribute), and thus, security and context are connected to the Attribute, *c.f.*, Figure 4. In Figure 4, the connections to Attribute start from the Security concepts and Context concepts frames, which means that all of those concepts can contain measurable attributes. Examples of attributes are a key length from the security side and the number of smart space devices from the context side. Naturally each attribute contains its own measures.

Figure 4. Dependencies of security and context ontologies.

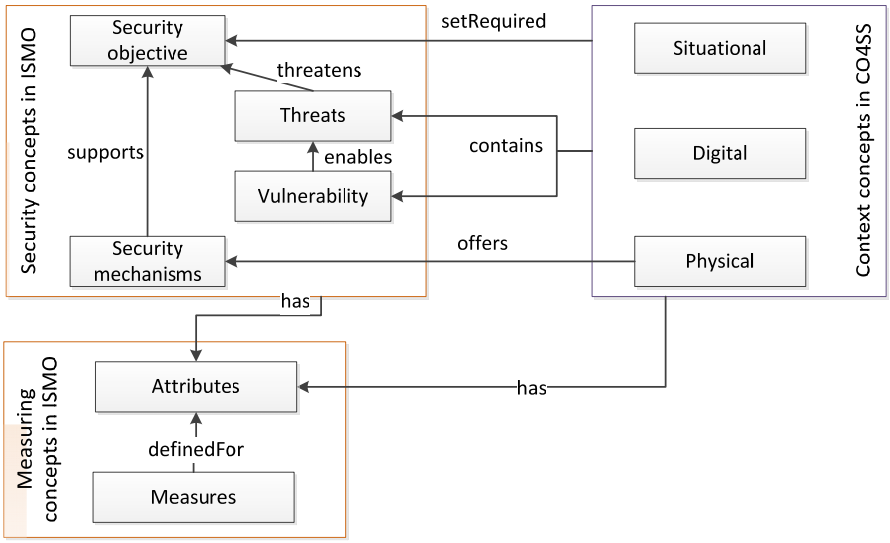


Figure 3 shows that knowledge from ontologies is utilized at design-time and runtime alike. At design-time, the software architect implements a set of monitoring probes, which require knowledge of the measures. Moreover, the architect searches which security mechanisms to implement from ISMO. Additional details of the design-time use of ISMO are presented in [38,53]. At runtime, the SSA automatically utilizes knowledge from the ISMO. The application has to know what measures to

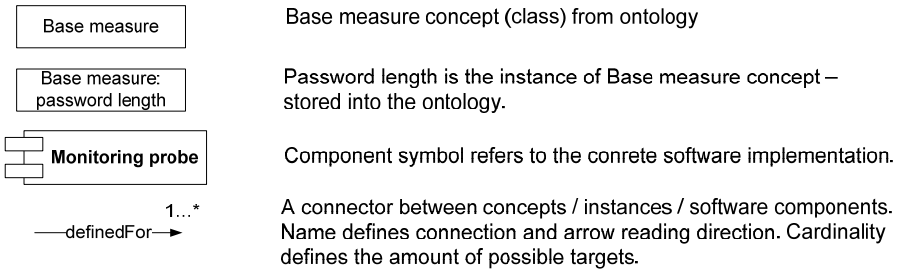
utilize, how to analyze results, and finally, how to create an adaptation plan. These knowledge requirements are described in detail in the following sections.

Smart spaces create a need to update knowledge every now and then. There can be several reasons for updates: new security mechanisms appear, vulnerabilities are found, the threat landscape changes or the execution environment changes. The SSA has to be aware of these changes, which is ensured by up-to-date knowledge. Utilizing ontologies as the knowledge source offers an advantage from a knowledge updating and enhancement view point. ISMO and CO4SS are presented in an OWL format, and thus, updates can be made to ontologies without modifying SSAs, which only retrieve knowledge from ontologies.

3.2. Architecture for Security Adaptation

The security-adaptation approach follows the MAPE model. The Monitor phase collects information by means of monitoring probes. The Analyze phase calculates the security level achieved, reasons the required security level, and calls the Plan phase if the required securities are not achieved. The Plan phase creates a plan to adapt, and finally, the Execute phase enforces the adaptation plan. The following descriptions of these phases show, how they utilize knowledge from the Conceptual interoperability level. Figure 5 presents a legend for figures used in the next sections.

Figure 5. Symbol definitions.



3.2.1. Monitor

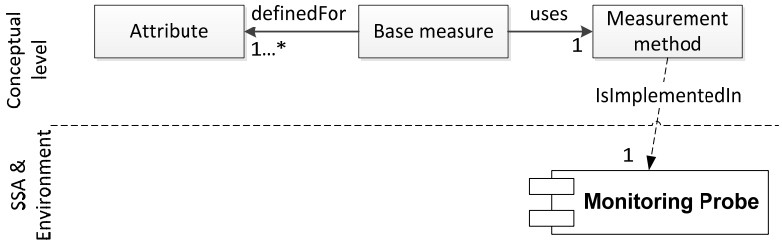
The Monitor phase gathers information from the environment and the SSA itself. The purpose of monitoring is to collect those small information pieces, which are then utilized to reveal an adaptation need. The target of monitoring can be at any level—from a low-level network technology to a high-level description of the user situation. These monitoring targets are called attributes—the encryption key length and the number of unknown devices in the smart space are examples of monitored attributes. Monitoring utilizes security measures, and thus, the measuring related terminology is utilized in the Monitoring phase. The Monitor phase uses Base measures to collect raw data by means of Monitoring probes. It is notable that Figure 4 presents the Measure concept, and the Base measure is one subclass of the Measure.

At design time, the software architect implements Monitoring probes into the SSA and environment based on the Measurement method descriptions from the ISMO, *c.f.*, Figure 6. If the architect creates a new Base measure and Monitoring probe, knowledge about what attribute the probe measures is added

into the ISMO. The Monitoring probe is a concrete code snippet, which is able to observe the particular attribute, e.g., by retrieving a key length from the utilized encryption library. In other words, the Monitoring probe is the implementation of the Measurement method. Each Monitoring probe is intended to observe only one attribute from the environment or SSA. Therefore, implemented probes can be easily reused.

Figure 6 depicts knowledge from the ISMO, *i.e.*, the Conceptual level, and its relation to the SSA and the environment in the Monitoring phase. The SSA and environment contain several attributes and each attribute has its own Monitoring probe implementation. At runtime, the ISMO provides knowledge about useful Base measures. The environment contains several probes but only a certain set is needed in each particular situation. For instance, in a situation where communication integrity is not needed it is useless to utilize integrity-related Base measures. Hence, the knowledge from the ISMO reveals which probes to use. Consequently, it is vital that the ISMO contains the connections depicted in Figure 6 because that information shows what attribute each probe is able to measure.

Figure 6. Knowledge from the Information Security Measuring Ontology (ISMO) for the Monitoring phase.



3.2.2. Analyze

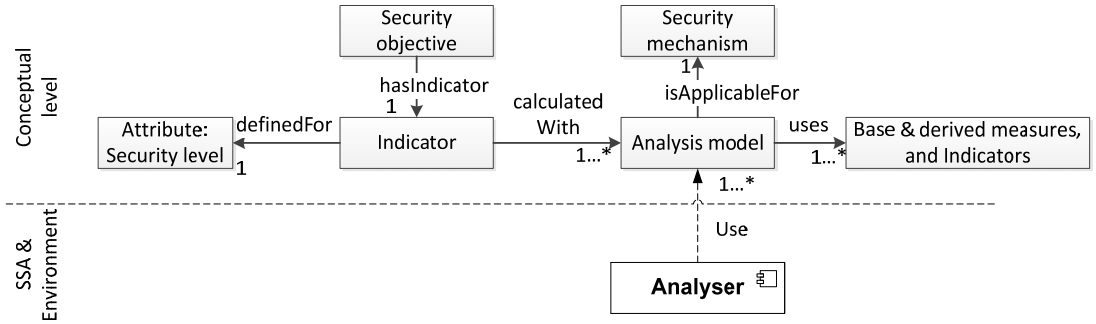
The Analyze phase reveals the current security levels for the required security objectives and decides which levels are sufficient for the current situation. Hence, the Analyze phase combines the Base measure results from the Monitoring phase to Indicators. The Indicator presents the security level of the security objective in that particular smart space situation. In other words, Base measures indicate that some changes have occurred in the smart space, whereas, the Analyze phase reveals the consequences of these changes, *i.e.*, it builds a meaning for the information in that situation.

The ISMO contains Derived measures and Indicators, which combine the Base measure results. Base measures, Derived measures and Indicators are sub-classes of the Measure concept, depicted in Figure 4. The ISMO follows a terminology defined in [26], and thus, Derived measures use the Measurement function and Indicators use Analysis models to perform the combining process. The Measurement function can be a simple mathematical operation, e.g., base_measure_1 + base_measure_2. In contrast, Analysis models contain more complex structures including conditional clauses and Boolean operations. For instance, the result from the Analysis model can be the integrity level of communication.

Figure 7 shows concepts for the Analyze phase from the ISMO. Each indicator has Analysis models, which use Base measures, Derived measures and Indicators. Every Security objective has its

own Indicator—e.g., authentication is a security objective, which has an Indicator called the authentication level. However, the Indicator is able to use several Analysis models, depending on the situation in hand. For example, a different Analysis model has to be used when authentication is based on fingerprints, passwords or a multi-factor authentication mechanism. In the multi-factor authentication case, the Analysis model, which combines analysis models from a single mechanism, is applied. Therefore, each Analysis model has a property that binds it to the security mechanism—from the above example, multi-factor authentication is seen as an individual security mechanism. Similarly as the Base measures, Indicators are also defined for Attributes. At design-time, the architect brings the Analyzer component into the SSA and the Analyzer component searches knowledge from the Analysis models to calculate security-level indicators for the SSA at runtime.

Figure 7. The concepts from the ISMO for the Analyze phase.

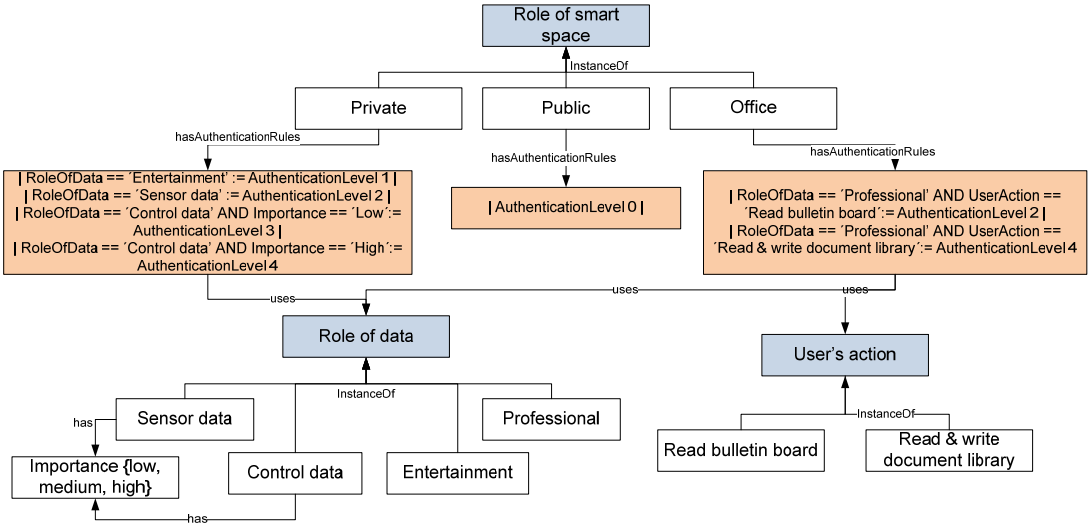


The security level indicators are compared to the required security levels. As depicted in Figure 4, the context information sets the required security objectives and levels. In [40], we defined the context concepts, which affect the required security as follows: (i) The user's role in the smart space. (ii) The actions performed in the smart space. (iii) The role and importance of the data. (iv) The role of the smart space. Furthermore, user preferences are able to affect the required security level.

Figure 8 shows the context information rules that define the required authentication level for different situations. However, the rule sets are an example—not the extensive rule sets to define authentication requirements. In these rules, the role of the smart space, *i.e.*, Private, Public or Office, constitutes the main categorization between the rule sets. In the private smart space, e.g., home smart space, the role of the data and its importance define how strong authentication has to be used. Thus, in a situation where the user consumes entertainment data—like news—authentication level 1 is sufficient. However, in a situation where the high importance level control information is handled, e.g., a home alarm system, authentication level 4 is required. In contrast, in public smart spaces, like freely available smart city services, authentication is not required at all. Finally, in an office smart space the user's action is also taken into account in the rules. Similar context-based rule sets can be defined for other security objectives, *i.e.*, confidentiality, integrity, availability, *etc*. Based on these rules, the SSA is aware of the required security objectives and levels in different situations.

The final step in the Analyze phase is to compare the Indicator results to the required security objectives and levels. If some indicators show that the required security level is not achieved, the Planning phase will be triggered.

Figure 8. An example of context information to define security objectives and levels.



As a whole, the Analyze phase requires a lot of knowledge, which is available from the ontologies. Based on the knowledge, the SSA builds an individual view point of the environment and achieved security. In order to produce the correct results, knowledge maintenance is important. Analysis models and the rule sets for the definition of requirements can be added and updated without modifying the application logic. The utilization of ontologies makes this flexibility possible, *i.e.*, the knowledge is not hard coded inside the application.

3.2.3. Plan

The Plan phase decides how to adapt the SSA when the required securities are not achieved. The Plan phase uses the results from the Analyze phase as input information, *i.e.*, (i) the unfulfilled security objective. (ii) The current and required security level. (iii) The used security mechanism. Furthermore, ontologies offer knowledge to make an adaptation plan. In some situations, only certain security mechanisms are supported, and thus, the Plan phase has to take these restrictions into account.

We have recognized three alternative ways to create the adaptation plan:

(1) The SSA utilizes pre-defined configuration alternatives.

(2) The SSA searches alternative security mechanisms or individual attributes to adapt, based on knowledge from the ISMO.

(3) The SSA asks the user how to proceed.

The first one is the simplest case. At design-time, the architect implements configuration alternatives inside the SSA. Thus, the Plan phase selects one of these pre-defined configurations at runtime. Naturally, dynamism is restricted in this alternative. However, this is enough for simple

devices and applications. As an example, for a situation where the required level of communication confidentiality is not achieved, the pre-defined configuration can be *"Start to use the TLS connection"*.

The second planning alternative changes the security mechanism or adapts individual attributes. When changing the whole security mechanism, the Planner component searches alternative mechanisms from the ISMO. In the ISMO, each security mechanism contains a link to the supported security objectives. For the required security objective, the ISMO can contain several security mechanisms. However, it is probable that only few of those are applicable in the current situation. For instance, the user's device supports fingerprint authentication but the smart space infrastructure does not offer this possibility. Hence, the adaptation plan has to take into account these restrictions from the context information. Alternatively, the Planner component can adapt individual attributes. To achieve this, the SSA searches the causes for the current security level by means of the same Analysis model, which showed that the adaptation was needed. In other words, the Analysis model is used to search Attributes, which have affected the current security level. Therefore, the SSA knows which attribute to adapt in order to affect the security level. Figure 9 illustrates this alternative—colored rectangles refer to the concepts presented in Figure 4. The Analysis model uses Base measures to observe Attributes. If the Attribute can be adapted its *adaptableWith* property shows an action for how to adapt the Attribute. The smart space may contain attributes that affect the achieved security level but all of these cannot be adapted. For instance, if the smart space contains an external threat that cannot be removed other attributes have to be adapted to mitigate threat effects. Finally, the Planner component decides on the adapted Attribute and the Action to be performed. To make this decision the Planner component may utilize goal, constraint or utility function based decision-making. At the moment, our approach is clearly goal orientated, *i.e.*, the context sets the required security objectives and levels (goals) and the purpose of the planning is to find a configuration that satisfies the goal. However, the decision-making algorithm is out of the scope of this paper but the architecture does not restrict decision-making algorithms utilized internally in the Planner component. When the Planner has to take trade-offs into account more sophisticated decision making will be needed, for instance for utility functions.

The third case is for a situation where the SSA is not able to create an enforceable adaptation plan. The following reasons can lead to this alternative: (i) The required knowledge is not defined in the ontologies. (ii) Knowledge is available but creating an adaptation plan would consume too many resources. Therefore, the only way to proceed is to give a warning message to the user and ask for instructions on how to continue.

From these alternatives, the second one is the most dynamic and autonomous without hard-coded adaptation plans. Hence, it is the preferred alternative.

3.2.4. Execute

The final step in the adaptation is the Execute phase, where the created adaptation plan is executed. In other words, it is the straightforward realization of the adaptation plan. At design-time, the software architect has to design variation points inside the SSA. The variation point ensures that it is possible to adapt security mechanisms and attributes at runtime. Figure 9 contains the Action to adapt concept. From the Execute phase point of of view the action is a component, which modifies the related

Attribute. Figure 10 shows a runtime situation, where the Planner component calls the Executor and offers the adaptation plan. Based on the adaptation plan the Executor signals an implementation component.

Figure 9. Planning using an Analysis Model.

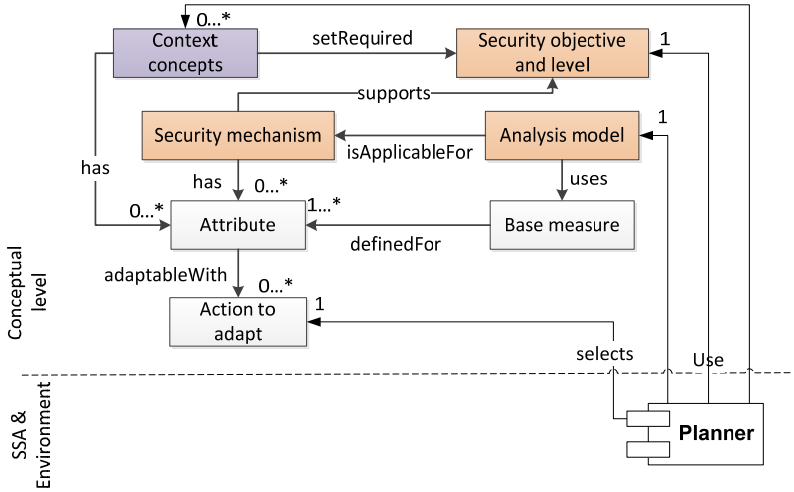
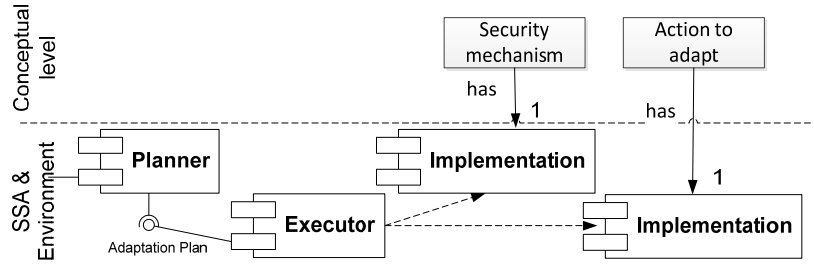


Figure 10. Execute phase.



The execution can affect different layers in a device where the SSA is running. Moreover, the execution can indirectly affect the whole smart space infrastructure. For example, when the SSA adapts a communication protocol the application has to establish a new connection to the smart space infrastructure by using new parameters or a new mechanism. The execute phase can, for example, control how information can be shared by defining security policies according to the Security Control Model described in the next section.

3.3. Runtime Security Control Model

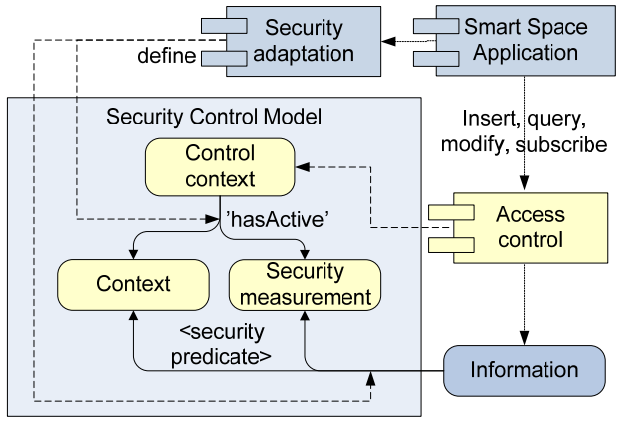
Security knowledge in the conceptual interoperability level provides a means to present security policies, which control the behavior of smart space devices and applications. However, analyzing and planning access-control decisions at runtime, when information is queried and modified, can be

computationally costly. In smart spaces the information is shared using SIBs, which are unaware of applications' conceptual policies and hence unable to enforce these policies. SIBs can be assumed to be aware only of a minimal set of standard security primitives, which are associated to information elements instead of the meaning of this information. In addition, as smart space devices may have limited computing capabilities, solutions based on cryptography are often unfeasible. Therefore, efficient solutions are needed to protect information sharing and to control information access in a fine-grained manner at the level of semantic data.

This subsection generalizes and formalizes our previously presented RDF access control approach [51] into a conceptual security control model. The control model has been verified with RDF but it can be applied to any information presentation system, which is based on subject-predicate-object triples. It specifies how access-control policies and security control information over resources are structured and presented. Runtime costs are minimized by requiring that each policy is presented with a single information triple. The security control model is based on context and security measurement concepts, which are used to authorize actions. Hence, the model can be applied efficiently and flexibly in various dynamic security-control situations.

Figure 11 depicts the security relationships in the security control model. The model has a relationship with three software components, presented in the top-right corner of the figure. *Smart space applications* insert, query, modify or subscribe to information resources. The *access control* component authorizes and controls these operations. Application specific *security adaptation* components administer the behavior of the access-control component. This administering is done by controlling the relationships between resources as specified by the security control model.

Figure 11. Runtime security-control model for smart spaces.



Each piece of information, *i.e.*, each *information* resource, can have a relationship with one or several *context and security measurement* resources. Each relationship presents one access control statement and is described using triplets in the form: "*Information, SecurityPredicate, Context/Measurement*". *Security predicates* are properties that define authorizing or accounting relationships for the security control. Predicates, which are to be used when authorizing transactions, are presented in Table 2. Predicates for accounting can be found in Table 2. *Context/measurement*

refers to any resource that the security adaptation component selects based on the ontologies and policy information from the conceptual level.

When the SSA queries or modifies information, only some contexts and measurements are active. The access control component uses resources, which are active for the application in the current situation. Active resources are found through the *control context* concept, which can be realized as a resource. Security adaptation components define which measurement and context resources are active with triplets: "*ControlContext, 'hasActive'*, *Context/Measurement*". Determination of what resources are active is a dynamic and constantly running process, which may involve different security-adaptation applications. ControlContext resources are fixed in the sense that the access control and security adaption components must know them. For instance, each SSA that is connected to a SIB and has an open communication socket may have a dedicated ControlContext resource. In this case, the active resources could be URIs representing the end-user's identity or security level. These URIs can be resolved and activated by the security adaptation component in the Monitor and Analyze phases, when the user authenticates.

3.3.1. Authorization Predicates

An important use case for the security-control model is authorization over resource access. Policy predicates enabling authorization are defined in Table 1. The granularity of the security-control model protects individual resources and also semantic relationships because of control over access to the resource properties. The security-control model supports the use of allow and disallow policies. Different policies can be used in conjunction to the set conditions of the authorizations (e.g., a user can access information but only if a contextual requirement is met). To prevent contradictory behavior due to the simultaneous use of allow and disallow policies, the proposed approach is that 'disable' policies override "allow" policies.

The security control model enables efficient runtime access control. An access-control component does not need to do heavy reasoning at the time applications are querying or modifying information, instead, security adaptation Analysis and Planning phases can be done in advance when adaptation-triggering events occur. The access control component needs to locate the relevant security relationships, presented with triples, between context or measurement resources and a target sources. When an SSA queries or modifies information, the access control component checks whether there are active policies allowing or denying the action.

When the amount of active and authorizing context and measurement resources is n, the access control component must do at most 2*n truth queries ("is there an allow or deny relationship between the active resource and the accessed resource?") to resolve the authorization of a transaction on a target. The access control component must also find active resources for each used control context resource. Implementations may further speed this up by keeping the list of control context specific active resources in the cached memory.

Table 1. Authorisation policy predicates.

Predicate	Description
GetAllowedFor	Authorizes reading a URI or literal value
SetAllowedFor	Authorizes modifying a URI or literal value
PropertyCreationAllowedFor	Authorizes adding a new URI or literal node under the URI node
PropertyRemovalAllowedFor	Authorizes the removal of a URI or literal node from the URI node
UseAsPropertyAllowedFor	Authorizes use of this node under other URI nodes
GetDisabledFor	Prevents reading a URI or literal value
SetDisabledFor	Prevents modifying a URI or literal value
PropertyCreationDisabledFor	Prevents adding a new URI or literal node under the URI node
PropertyRemovalDisabledFor	Prevents the removal of a URI or literal node from the URI node
UseAsPropertyDisabledFor	Prevents the use of this node under other URI nodes
IsAuthorisedBy	Sets a node under access control and specifies authority. There may be several authorities in one broker.

Table 2. Predicates for access control accounting.

Predicate	Description
HasBeenAuthoredBy	Identifies a resource's author
HasAddedPredicate	Identifies authors who have added predicates under the resource
IsSignedWith	Link to a signature proving authenticity and the origin of the resource
HasSecurityContext	Link to any security measurement or context resource which was active when the data was stored (needed to verify e.g. trustworthiness of data)
IsAuthorisedBy	Specifies the authority that controls security. If such relationship to a known security authority is missing, access can be directly authorized without any other checks.
CanBeMonitored	Allows or disallows logging (e.g. due to performance or privacy)
HasBeenReadBy	Identifies contexts (users) where data has been successfully queried
HadInvalidReadAttemptBy	Identifies contexts (users) with rejected read requests
HadInvalidWriteAttemptBy	Identifies contexts (users) who have made rejected write requests

3.3.2. Accounting Predicates

In addition to authorization, the security control model supports other real-time security control situations. Table 2 presents predicate definitions for access accounting activities, which are needed to determine the authenticity or trustworthiness of information. The table defines the relationships for accounting predicates, which are used to log access requests, both successful and unsuccessful. This

information is needed, e.g., when trying to detect malicious or harmful modifications and intrusions and when reasoning which nodes may have been potentially compromised due to harmful information. The table also lists *IsSignedWith* and *HasSecurityContext* predicates, which users can use to verify the authenticity and trustworthiness of information. Trustworthiness may depend on context or measurement, which were active when the information was stored.

4. Implementation of Adaptive Security

4.1. Case Description

The validation was based on a use case from the smart space pilot—called Seamless Usage of Multiple Smart Spaces (SUM-SS) [55]. The SUM-SS pilot combines four smart spaces, *i.e.*, smart personal space, smart home, smart office and smart city. An end user facilitates information from these smart spaces via his/her mobile phone, which constitutes his/her personal smart space. In the personal smart space, the user is able to store information from other smart spaces, like calendar information and documents from the office space. The home smart space offers capabilities to monitor energy consumption; control light and wall sockets and control home automation via the Lon network etc. The smart city offers public information and facilitates everyday life in an urban environment—for instance, by offering information on parking areas and traffic jams. Furthermore, mobile devices and televisions are able to consume entertainment content from a cloud, which is supported by the Cam4Home platform [56]. Consequently, the SUM-SS pilot opens up possibilities to select a smart space use case for the validation purposes. Hence, the use case selected for the validation purposes concentrates on illustrating the following issues:

1. The SSA running in an end user's mobile phone utilizes the adaptation loop to ensure an appropriate authentication level in different situations.
2. Security- and context-related knowledge is retrieved from ontologies.
3. RIBS controls access over shared information by using the security control model.

Use case description: The homeowner leaves the home in the morning, by car. During the drive she wants to check that the front door of her house is locked properly. The owner is able to check a lock status via her mobile phone without any additional authentication.

During the working day, a maintenance man arrives on the front door of the house and rings a doorbell. The doorbell sound is played in the owner's mobile phone and a video stream from the front door of the house is delivered. Hence, when she recognizes the maintenance man at the door, she is able to open the door remotely. However, her current authentication level is not strong enough for the remote door opening, and thus, re-authentication is requested.

After a while, the maintenance man is ready and leaves the house. The owner is informed and she locks the front door again. Now the time has passed and the authentication level is dropped. Nevertheless, re-authentication is not needed because the door can be locked with a lower authentication level.

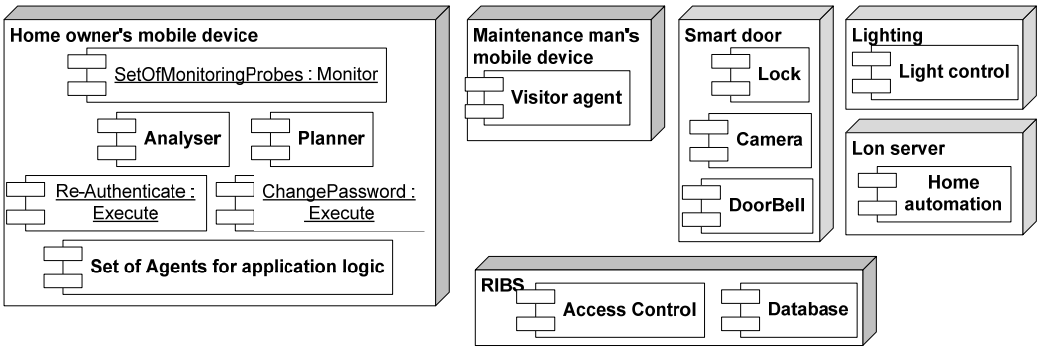
The home owner arrives home and opens the front door locally by utilizing the NFC (Near Field Communication) feature of her mobile phone. The mobile phone authenticates the user and then the mobile is authenticated through NFC. Inside the house, the owner adjusts the lighting—and the earlier

achieved authentication level is enough for these actions. After a while, the owner wants to turn down the heating system in the house. However, controlling the home automation system is a critical action and too much time has passed from the last authentication. Thus, authentication with a stronger authentication level is required.

4.2. Case Implementation

Figure 12 shows the deployment of the use case from the security adaptation viewpoint. The use case implements user authentication in an adaptive manner by using password-based authentication. Our previous demonstration utilized a gait-based authentication [57]. The construction contains six nodes, *i.e.*, homeowner's mobile device, RIBS, smart door, lighting, Lon server and the maintenance man's mobile device. The main actions in the use case are performed with the homeowner's mobile device Nokia C7. Hence, it contains application logic agents to retrieve and insert information into the RIBS. These agents are implemented with Qt C++. In this case, the adaptation will be performed from the homeowner's viewpoint, and thus, adaptation-related components are located in her device. The smart door node contains lock, camera and doorbell agents, which offer related functionalities. Similarly, the lighting node and the Lon server node contain agents to utilize those devices. The RIBS is executed inside a WLAN access point—in order to offer a good connectivity. The RIBS supports the Transport Layer Security (TLS) protocol [58] to secure communication between agents and the RIBS. The last node is the maintenance man's mobile device that contains a visitor agent. The visitor agent makes it possible to ring the doorbell that is available publicly from the home smart space.

Figure 12. The deployment of the use case.

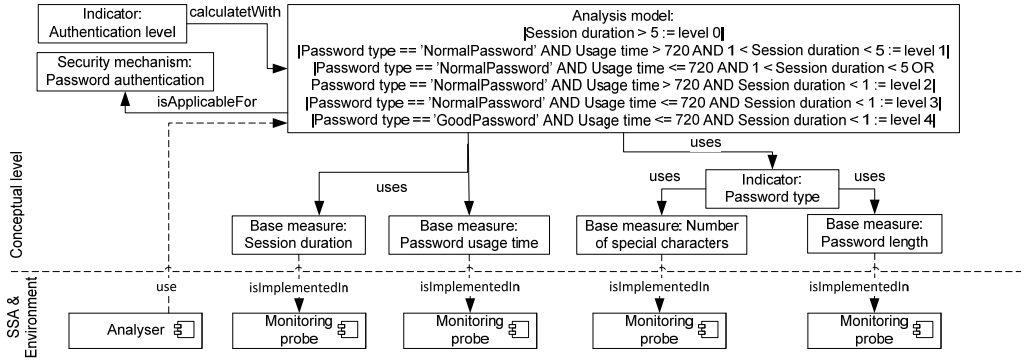


Monitoring probes are components that implement measurement methods for base measures (*c.f.*, Figure 6). For password-based authentication the use case utilizes the following base measures: (1) Password length. (2) The number of special characters in the password. (3) Password usage time. (4) Session duration. The change of these base measures is informed to the Analyze component. Naturally, Base measures 1 and 2 change when the password is changed. In contrast, Base measures 3 and 4 change constantly, and thus, the changed values are informed to the Analyze component at a certain intervals.

The Analyze component combines the monitoring results to authentication level indicator by means of the Analysis model (*c.f.*, Figure 7). The Analyzer component retrieves the right analysis model from

the ISMO. Analysis models are described by a natural language, which combines English and Boolean algebra. We made this decision in our previous work [38] in order to facilitate the preparation of Analysis models. Thus, Analysis models can be modified and added without experience of ontology query languages. The Analysis model utilized in this case is presented in Figure 13. The analysis model uses four base measures—either directly or via the Password type indicator—as depicted in the figure. Moreover, the figure presents Monitoring probes, which implement measurement methods for base measures.

Figure 13. Analysis model used in the use case.



The above-presented Analysis model produces the achieved authentication level. Moreover, the Analyze component analyzes the current situation in order to decide the required authentication level. For this purpose, rules presented in Figure 8 are utilized.

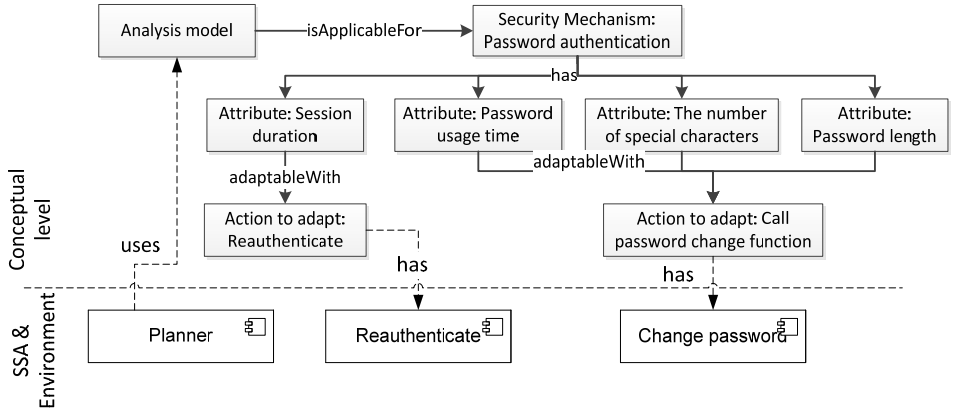
Table 3 summarizes user actions, situations and the achieved and required authentication levels. The achieved authentication level means the level before adaptation. Hence, Actions 1, 3 and 5 do not require adaptation because the achieved level is higher than the required level. Other actions require adaptation and the Planner component is called.

The Plan component selects how the adaptation is performed. In this case, the Planner component searches the causes of the current security level from the analysis model (*c.f.*, Figure 9). Figure 14 lists the attributes of password authentication. In the ISMO base measures from Figure 13 are connected to these attributes by means of a *definedFor* property. Now there are two possible ways to adapt. Firstly, the re-authentication of the user affects the session duration attribute. Secondly, calling the change password function affects password usage time—and depending on a new password—the length and number of special characters attributes. Based on this knowledge, the Planner component selects an appropriate action to adapt. Table 3 listed user actions. Action numbers 2 and 4 are handled with re-authenticate adaptation. The user action number 6 leads to the change-password adaptation action because the one-month usage time, *i.e.*, 720 h, boundary has been exceeded.

Table 3. Achieved and required authentication levels in different situations.

Action	Situation and Input Information for the Analysis Model	Achieved Auth. Level	Required Auth. Level
1. Check the lock status	Check the lock status Normal password selected 708 h ago. Session duration: 2 h	2	0
2. Open the front door	Remote opening Normal password selected 710 h ago. Session duration: 4 h	2	3
3. Lock the front door	Remote locking Normal password selected 712 h ago. Session duration: 2 h	2	1
4. Open the front door	Local opening Normal password selected 719 h ago. Session duration: 7 h	0	3
5. Modify lighting	Local modification Normal password selected 719 h ago. Session duration: 0.02 h	3	1
6. Modify home automation	Local modification Normal password selected 721 h ago. Session duration: 2 h	1	2

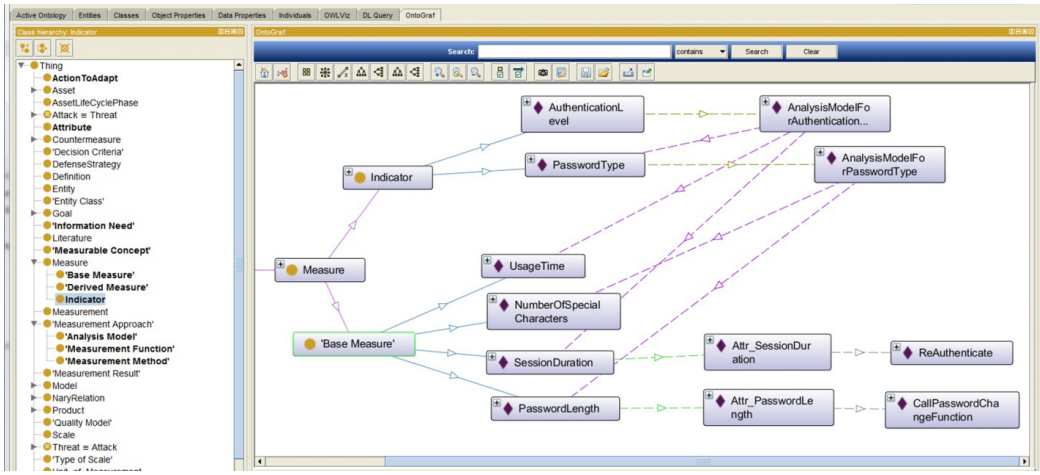
Figure 14. Actions to adapt different parameters.



Lastly, the Execute component performs the created adaptation plan. Both adaptation actions also have an effect outside of the homeowner's device: It requires re-authentication into the RIBS or calling the password-change function from the RIBS. However, in both cases the initiative for these actions is made in the Execute component inside the homeowner's device.

Figure 15 shows the screenshot of the ISMO in the Protégé ontology tool. The screenshot contains concepts instantiated for the use case purposes. The ISMO is also available in web: https://sofia-community.com/projects/sontologies/—needs registration. The page contains ISMO and links to imported ontologies.

Figure 15. ISMO in Protégé.



The RIBS is responsible for enforcing access control over brokered information. It enforces that only authenticated and authorized users can insert and modify information. The authorization checks follow the runtime security-control model, as illustrated in Figure 16. The agents in the homeowner's terminal are responsible for administering the authorization policies, which are stored in the RIBS for each RDF resource. When a user is authenticated, appropriate context and measurement resources are activated. In the use case, the maintenance man is mapped to a visitor context and the homeowner is mapped to a resource, which represents owner's identity. Further, all users are mapped to security measurement resources, which describe the authentication level. When users query or modify information, the SIB checks whether these active RDF resources authorize access to the requested resources. All authenticated users are given access to non-critical information inside the home e.g., the lighting. The homeowner has access to every piece of information. However, access to the most critical information requires that the owner have a sufficient authentication level.

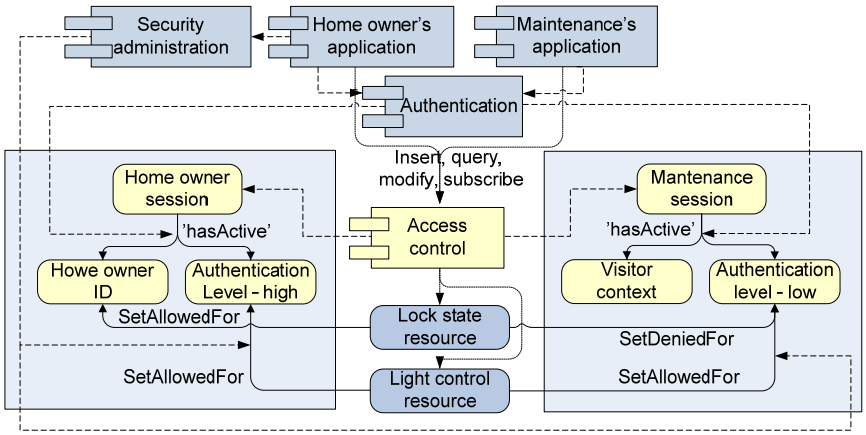
The RIBS has been optimized to provide fast and low-power consuming information access. The implementation indexes all incoming RDF data, and thus, enables RDF URIs as well as literals to be directly addressed. Relationship information is stored to a three-dimensional (subject-predicate-object) array, *i.e.*, to a bit cube. As security policies are presented with a single bit, which is either on or off, they can be quickly checked and the amount of required memory will not increase even when the security configuration becomes more complex.

4.3. Lessons Learned

Implementing and performing these use cases showed several advantages from the presented approach. Firstly, all knowledge is retrieved from ontologies. Thus, content from the analysis model (*c.f.*, Figure 13) and rules for setting the required authentication levels (*c.f.*, Figure 8) do not need to be coded into the SSA. Hence, knowledge can be modified and extended without coding work. Secondly, adaptive user authentication enhanced usability. However, as the authentication level decreased the user was able to perform Action 3 (see Table 3). In contrast, using a static authentication requires that

all actions be set as equally critical, which in turn means that all actions require the highest authentication level. Thirdly, from the implementation view point the presented adaptation loop clearly defined the required components. In addition, knowledge—required in each component—was defined explicitly. Therefore, the approach description can act as an implementation guideline. Finally, the security-control model in the RIBS supported flexible access control. The access control information is described in the RIBS as RDF information, and thus, separated access-control lists are not needed.

Figure 16. Examples of authorizing relations in the runtime security-control model.



However, improvement ideas are also recognized. The first consideration relates to the selected security objective—in the use case user authentication was in the focal point. Supporting other security objectives does not require changes to the adaptation approach and utilization of knowledge from the ontologies. Nevertheless, the content of the ontologies has to be extended with new knowledge related to security objectives. This knowledge does not need to be created from scratch but the existing knowledge has to be described in a machine-readable form by means of the ontologies. The second issue relates to the definition of the Analysis model. During the case implementation it was noticed that defining the Analysis model is a complex and time-consuming task—although only three variables were used. Thus, automation is needed in the future in order to define extensive analysis models.

5. Discussion

5.1. The Advantages of the Approach

Achieving security in smart spaces requires dynamic security solutions. The presented adaptation approach for security ensures that the achieved security corresponds to each situation faced in the smart space. Further, the introduced security-control model ensures the appropriate information access in changing smart spaces. Thus, the introduced security-adaptation approach offers several advantages:

Firstly, the approach is generic—security objectives and mechanisms can be freely selected. Hence, the approach is able to offer adaptation for one security objective, or alternatively, a set of security

objectives can be implemented in an adaptive manner. Similarly, the adaptation approach is independent on the used security mechanism.

Secondly, the adaptation approach is based on knowledge from ontologies, which cover the knowledge part of the MAPE-K model. This separation of concerns, *i.e.*, separation of generic knowledge from application logic, improves reusability. The knowledge can be updated and extended easily and quickly without modifying the SSA. Moreover, the ontologies ensure that there is a uniform way to present security terminology.

Thirdly, the commonly known MAPE model ensures that the required components are clearly defined, *i.e.*, Monitoring, Analyzing, Planning and Executing. Clearly defined components support reusability, which facilitates the architects' work. The Monitoring utilizes security measures and the Analysis utilizes extendable analysis models. In each phase the required knowledge is retrieved from the ontologies.

Fourthly, smart spaces consist of devices and applications that can enter and leave the space at any time. Various situations require dynamic access-control policies that are enforced automatically. The security-control model provides an approach for enforcing the access control for different situations. The model is expressive but still straightforward, and hence, more suitable for runtime enforcement than the previous RDF level access-control proposals.

5.2. The Challenges of the Approach and Future Research

Firstly, the use cases showed that creating analysis models is a complex task. Therefore, a tool, or at least guidelines, for the analysis model definition is needed in the future. The tool has to present the possible variables and their value ranges. In addition, the tool has to check that contradicting analysis models are not created.

Secondly, in the future mutual relationships between security objectives have to be taken into account during the Analyze and Plan phases. The ISMO already contains basic connections between security objectives, e.g., authorization demands identification. Therefore, analysis models have to notice these dependencies when recognizing adaptation needs. Similarly, the Plan phase is able to utilize this knowledge when searching applicable adaptation actions. For instance, the need to adapt authorization causes the utilized identification scheme to change. Furthermore, trade-offs and dependencies on other qualities like performance, reliability and usability are topics for future research.

Thirdly, the performance cost of the approach is a natural question. However, end users were not able to recognize decreased performance during the case study. It is clear that the performance overhead can be noticed if a huge amount of base measures and complex analysis models are used. Therefore, it is important to adjust the analysis models and the number of base measures for device resources. The performance penalty caused by the RDF access control depends on the amount of requested RDF resources. In a performance test case with the RIBS implementation, the average request times were around one per cent longer when compared to a case where all requests were authorized without any checks [51].

The last issue relates to the security of the adaptation approach. It is possible that an attacker may try to modify some parts of the adaptation loop in order to attack the smart space. For instance,

manipulating the Monitoring or Analyzing parts might mean that a decreased security level is not recognized. Or alternatively, the Plan phase may create an inappropriate adaptation plan, which is advantageous for the attacker. Hence, it is extremely important to protect these components and ensure the authenticity and integrity of knowledge.

5.3. The Maturity of the Approach

The approach was developed during a three-year research project. The development was performed incrementally by utilizing different use cases and smart space set-ups. In previous cases, we have experienced different adaptation ideas and worked on different phases of security adaptation. Table 4 summarizes the previous validation cases.

Table 4. Previous validation cases.

Validation Case	Description
Risk-based security adaptation in a greenhouse [41,57].	A greenhouse with a shopping area constitutes a public smart space. In the smart space threats increase the risk levels and security mechanisms decrease risks. Hence, the monitoring concentrates on recognizing threats. In this case, confidentiality and integrity were considered. Furthermore, users authenticated by means of gait information identified from the measurements of acceleration sensors inside the mobile phone.
Adaptive user authentication [38].	The first case that utilized knowledge from the ISMO. It adapts user authentication by monitoring authentication-related measures: password length, age, variation of characters and session duration. Important information was available only when an acceptable authentication level was reached. The user's re-authentication was requested when the session duration was exceeded.
Role- and popularity-based access-control simulations [51,59].	Controlling access to information according to the user's role or popularity of information. Popularity is a measure that indicates how many readers or how many authors an RDF resource has. These adaptation cases were simulated with the smodels logic solver.
Adding new knowledge into the ISMO [54].	The paper and related case example showed how easily knowledge in the ISMO can be extended. Moreover, design steps to develop adaptive security were presented.

6. Conclusions

In smart spaces, it is not possible to take all the security requirements into account at design-time. Hence, this paper presented a self-adaptation approach for smart space security. The presented approach contains an adaptation loop the Monitor, Analyze, Plan, and Execute model—in a clearly defined form. The monitor phase utilizes monitoring probes to observe security-relevant attributes from the smart space and smart space application. The monitored results are analyzed in order to reveal if the required security is not achieved. The Plan phase creates an adaptation plan, which will be enforced in the Execute phase. The adaptation approach requires a lot of knowledge, which is retrieved from the ontologies. The utilization of ontologies ensures a flexible and extensible way to manage and

use knowledge in machine-readable form. The ISMO provides security- and measuring-related knowledge and CO4SS offers context knowledge. For access control, the security-control model was presented, which utilizes context information and provides dynamic access control. Hence, the security-control model provides a flexible and efficient mechanism to control information sharing in smart spaces.

The presented approach was validated by means of a use case. The use case illustrated (i) all the phases of the adaptation loop, (ii) how ontologies offer knowledge for adaptation at runtime, (iii) that access control enforced the semantic information. The advantages of the presented approach are evident. Firstly, the approach is independent of security objectives and mechanisms. Second, the approach provides a reusable architecture to develop adaptive security applicable for different kinds of smart spaces. Thirdly, the components for the security adaptation are clearly defined, which help in adopting the approach. Finally, the utilization of ontologies ensures that knowledge can be updated and extended easily.

In the future, new and wider analysis models are needed. Thus, a tool will be developed for defining a wider set of analysis models for various security objectives.

Acknowledgments

This work has been carried out in the SOFIA ARTEMIS project (2009-2011) and SASER-Siegfried Celtic-Plus project (2012-2015) funded by Tekes (the Finnish Funding Agency for Technology and Innovation), VTT Technical Research Centre of Finland and the European Commission.

Conflict of Interests

The authors declare no conflict of interests.

References

1. Conti, M.; Das, S.K.; Bisdikian, C.; Kumar, M.; Ni, L.M.; Passarella, A.; Roussos, G.; Tröster, G.; Tsudik, G.; Zambonelli, F. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence. *Pervasive Mob. Comput.* **2012**, *8*, 2–21.
2. Elkhodary, A.; Whittle, J. A Survey of Approaches to Adaptive Application Security. In *Proceedings of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Minneapolis, USA, 20-26 May, 2007; pp. 16–23.
3. Yuan, E.; Malek, S. A taxonomy and survey of self-protecting software systems. In *Proceedings of the IEEE Software Engineering for Adaptive and Self-Managing Systems*, Zürich, Switzerland, 4-5 June, 2012; pp. 109–118.
4. Cook, D.J.; Das, S.K. How smart are our environments? An updated look at the state of the art. *Pervasive Mob. Comput.* **2007**, *3*, 53–73.
5. Ovaska, E.; Salmon Cinotti, T.; Toninelli, A. Design principles and practices of interoperable smart spaces. In *Advanced Design Approaches to Emerging Software Systems: Principles, Methodologies, and Tools*; Liu, X., Li, Y., Eds.; IGI Global, 2011; pp. 18–47.

6. Pantsar-Syväniemi, S.; Purhonen, A.; Ovaska, E.; Kuusijärvi, J.; Evesti, A. Situation-Based and Self-Adaptive Applications for Smart Environment. *J. Ambient Intelligence Smart Environ.* **2012**, *4*, 491–516.
7. Honkola, J.; Laine, H.; Brown, R.; Tyrkkö, O. Smart–M3 information sharing platform. In *Proceedings of the IEEE Symposion on Computers and Communications*, Riccione, Italy, 22–25 June, 2010; pp. 1041–1046.
8. RDF Primer. 2004. Available online: http://www.w3.org/TR/rdf-primer/ (accessed on 23 November 2012).
9. SOFIA Smart Objects For Intelligent Applications. www.sofia-project.eu, 2012. Accessed 23 November 2012.
10. Suomalainen, J.; Hyttinen, P.; Tarvainen, P. Secure information sharing between heterogeneous embedded devices. In *Proceedings of the 4th European Conference on Software Architecture: Companion Volume*, Copenhagen, Denmark, 23-26 August, ACM, 2010; pp. 205–212.
11. Kephart, J.O.; Chess, D.M. The vision of autonomic computing. *Computer* **2003**, *36*, 41–50.
12. Dobson, S.; Denazis, S.; Fernández, A.; Gaïti, D.; Gelenbe, E.; Massacci, F.; Nixon, P.; Saffre, F.; Schmidt, N.; Zambonelli, F. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.* **2006**, *1*, 223–259.
13. Salehie, M.; Tahvildari, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.* **2009**, *4*, 1–42.
14. Psaier, H.; Dustdar, S. A survey on self-healing systems: approaches and systems. *Computing* **2011**, 91, 43–73.
15. Huebscher, M.C.; McCann, J.A. A survey of autonomic computing-degrees, models, and applications. *ACM Comput. Surv.* **2008**, *40*, 1–28.
16. Matinlassi, M.; Niemelä, E. The impact of maintainability on component-based software systems. In *Proceedings of the 29th Euromicro Conference*, Belek-Antalya, Turkey, 3–5 September, IEEE, 2003; pp. 25–32.
17. Hashii, B.; Malabarba, S.; Pandey, R.; Bishop, M. Supporting reconfigurable security policies for mobile programs. *Computer Networks* **2000**, *33*, 77–93.
18. Hu, W.; Hiser, J.; Williams, D.; Filipi, A.; Davidson, J.W.; Evans, D.; Knight, J.C.; Nguyen-Tuong, A.; Rowanhill, J. Secure and practical defense against code-injection attacks using software dynamic translation. In *Proceedings of the 2nd international conference on Virtual execution environments*, Ottawa, Canada, 14-16 June, ACM, 2006; pp. 2–12.
19. Knight, J.C.; Strunk, E.A. Achieving critical system survivability through software architectures. In Architecting Dependable Systems II, Lemos R., Gacek C., Romanovsky A., Eds.; Springer: Berlin-Heidelberg, Germany, 2004; pp. 51–78.
20. Ryutov, T.; Zhou, L.; Neuman, C.; Leithead, T.; Seamons, K.E. Adaptive trust negotiation and access control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, Stockholm, Sweden, 1-3 June 2005; pp. 139–146.
21. Klenk, A.; Niedermayer, H.; Masekowsky, M.; Carle, G. An architecture for autonomic security adaptation. *Ann. Telecommun.* **2006**, *61*, 1066–1082.

22. Hulsebosch, R.; Bargh, M.; Lenzini, G.; Ebben, P.; Iacob, S. Context sensitive adaptive authentication. In *Smart Sensing and Context*; Kortuem, G., Finney, J., Lea, R., Sundramoorthy, V., Eds.; Springer: Berlin-Heidelberg, Germany, 2007; pp. 93–109.
23. Abie, H.; Savola, R.M.; Bigham, J.; Dattani, I.; Rotondi, D.; Da Bormida, G. Self-Healing and Secure Adaptive Messaging Middleware for Business-Critical Systems. *Int. J. Adv. Se.* **2010**, *3*, 34–51.
24. Savola, R.; Abie, H. Development of measurable security for a distributed messaging system. *Int. J. Adv. Se.* **2009**, *2*, 358–380.
25. Wang, C.; Wulf, W. A. Towards a Framework for Security Measurement. In *Proceedings of the 20th National Information Systems Security Conference*, Baltimore, Maryland, USA, October 1997; pp. 522–533.
26. García, F.; Bertoa, M.F.; Calero, C.; Vallecillo, A.; Ruíz, F.; Piattini, M.; Genero, M. Towards a consistent terminology for software measurement. *Inf. Softw. Technol.* **2006**, *48*, 631–644.
27. Haley, C.B.; Laney, R.; Moffett, J.D.; Nuseibeh, B. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Trans. Softw. Eng.* **2008**, *34*, 133–153.
28. Salehie, M.; Pasquale, L.; Omoronyia, I.; Ali, R.; Nuseibeh, B. Requirements-driven adaptive security: Protecting variable assets at runtime. In *Proceedings of the 20th International Requirements Engineering Conference (RE)*, Chicago, USA, 24–28 September, IEEE, 2012; pp. 111–120.
29. ISO/IEC 15408-1:2009 Standard. Common Criteria for Information Technology Security Evaluation – Part 1: Introduction and general model, International Organization of Standardization, 2009.
30. Sahinoglu, M. Security meter: a practical decision-tree model to quantify risk. *Security Privacy* **2005**, *3*, 18–24.
31. Zhou, J. Knowledge Dichotomy and Semantic Knowledge Management. In *Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web*, Jyväskylä, Finland, 25–27 August, Springer, USA, 2005; pp. 305–316.
32. Blanco, C.; Lasheras, J.; Valencia-García, R.; Fernández-Medina, E.; Toval, A.; Piattini, M. A systematic review and comparison of security ontologies. In *Proceedings of the 3rd International Conference on Availability, Security, and Reliability*, Barcelona, Spain, 4–7 March, IEEE, 2008; pp. 813–820.
33. Evesti, A.; Ovaska, E.; Savola, R. From security modelling to run-time security monitoring. In *Proceedings of the European Workshop on Security in Model Driven Architecture, CTIT Centre for Telematics and Information Technology*, Enchede, Netherlands, 23–26 June, 2009; pp. 33–41.
34. Kim, A.; Luo, J.; Kang, M. Security Ontology for annotating resources. In *Proceedings of the On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, Agia Napa, Cyprus, 31 October–4 November, Springer: Berlin-Heidelberg, Germany, 2005; pp. 1483-1499.
35. Denker, G.; Kagal, L.; Finin, T. Security in the Semantic Web using OWL. *Inform. Sec. Tech. Rep.* **2005**, *10*, 51–58.

36. Savolainen, P.; Niemelä, E.; Savola, R. A taxonomy of information security for service centric systems. In *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, Lübeck, Germany, 27–31 August, IEEE, 2007; pp. 5–12.
37. Herzog, A.; Shahmehri, N.; Duma, C. An ontology of information security. *J. Inform. Sec. Privacy* **2007**, *1*, 1–23.
38. Evesti, A.; Savola, R.; Ovaska, E.; Kuusijärvi, J. The Design, Instantiation, and Usage of Information Security Measuring Ontology. In *Proceedings of the 2nd International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*, Budapest, Hungary, 17-22 April, IARIA, 2011; pp. 1–9.
39. Pantsar-Syväniemi, S.; Kuusijärvi, J.; Ovaska, E. Supporting Situation-awareness in Smart Spaces. In *Proceedings of the International Workshops, S3E, HWTS, Doctoral Colloquium, Held in Conjunction with GPC 2011*, Oulu, Finland, 11–13 May 2011, Springer: Berlin-Heidelberg, Germany, 2012; pp. 14–23.
40. Evesti, A.; Pantsar-Syväniemi, S. Towards micro architecture for security adaptation. In *Proceedings of the 4th European Conference on Software Architecture, Companion Volume*, Copenhagen, Denmark, 23–26 August, ACM, 2010; pp. 181–188.
41. Evesti, A.; Ovaska, E. Ontology-Based Security Adaptation at Run-Time. In *Proceedings of the 4th International Conference on Self-Adaptive and Self-Organizing Systems*, Budapest, Hungary, 27 September–1 October 2010, IEEE, 2010; pp. 204–212.
42. Dietzold, S.; Auer, S. Access control on RDF triple stores from a semantic wiki perspective. In *Proceedings of the Scripting for the Semantic Web Workshop at 3rd European Semantic Web Conference*, Budva, Montenegro, 11–14 June 2006, CEUR Workshop Proceedings, 2006; pp. 1–9.
43. D'Elia, A.; Honkola, J.; Manzaroli, D.; Salmon Cinotti, T. Access Control at Triple Level: Specification and Enforcement of a Simple RDF Model to Support Concurrent Applications in Smart Environments. In *Proceedings of the 11th International Conference, NEW2AN 2011, and 4th Conference on Smart Spaces, ruSMART 2011*, St. Petersburg, Russia, 22–25 August 2011, Springer: Berlin-Heidelberg, Germany, 2011; pp. 63–74.
44. Reddivari, P.; Finin, T.; Joshi, A. Policy-based access control for an RDF store. In *Proceedings of the Policy Management for the Web*, Chiba, Japan, 10–14 May 2005; pp. 78–81.
45. Jain, A.; Farkas, C. Secure resource description framework: an access control model. In *Proceedings of the 11th symposium on Access control models and technologies*, Lake Tahoe, CA, USA, 7–9 June 2006, ACM, 2006, pp. 121–129.
46. Flouris, G.; Fundulaki, I.; Michou, M.; Antoniou, G. Controlling access to RDF graphs. In *Proceedings of the Future Internet - FIS 2010*, Berlin, Germany, 20–22 September 2010, Springer: Berlin-Heidelberg, Germany, 2010, pp. 107–117.
47. Kim, J.; Jung, K.; Park, S. An Introduction to Authorization Conflict Problem in RDF Access Control. In *Proceedings of the Knowledge-Based Intelligent Information and Engineering Systems*, Zagreb, Croatia, 3-5 September 2008, Springer: Berlin-Heidelberg, Germany, 2008; pp. 583–592.
48. Cho, E.; Kim, Y.; Hong, M.; Cho, W. Fine-Grained View-Based Access Control for RDF Cloaking. In *Proceedings of the 9th International Conference on Computer and Information Technology*, Xiamen, China, 11-14 October 2009, IEEE, 2009; pp. 336–341.

49. Bock, J.; Haase, P.; Ji, Q.; Volz, R. Benchmarking OWL reasoners. In *Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, Tenerife, Spain, CEUR Workshop Proceedings, 2008; pp. 1–15.
50. Dentler, K.; Cornet, R.; Ten Teije, A.; De Keizer, N. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web* **2011**, *2*, 71–87.
51. Suomalainen, J.; Hyttinen, P. Security Solutions for Smart Spaces. In *Proceedings of the 11th International Symposium on Applications and the Internet*, Munich, Germany, 18-21 July 2011, IEEE, 2011; pp. 297–302.
52. Niemelä, E.; Evesti, A.; Savolainen, P. Modeling quality attribute variability. In *Proceedings of the 3rd International Conference on Evaluation of Novel Approaches to Software Engineering*, Funchal, Portugal, 4-7 May 2008; pp. 169–176.
53. Ovaska, E.; Evesti, A.; Henttonen, K.; Palviainen, M.; Aho, P. Knowledge based quality-driven architecture design and evaluation. *Inf. Softw. Technol.* **2010**, *52*, 577–601.
54. Evesti, A.; Ovaska, E. Design Time Reliability Predictions for Supporting Runtime Security Measuring and Adaptation. In *Proceedings of the 3rd International Conference on Emerging Network Intelligence*, Lisbon, Portugal, 20–25 November 2011, IARIA, 2011; pp. 94–99.
55. Sofia Pilot Brochure. 2012. Available online: http://www.slideshare.net/sofiaproject/sofia-project-brochure-pilots-set (accessed on 23 November 2012).
56. Cam4Home Project. 2012. Available online: http://www.cam4home-itea.org/ (accessed on 8 May 2012).
57. Evesti, A.; Eteläperä, M.; Kiljander, J.; Kuusijärvi, J.; Purhonen, A.; Stenudd, S. Semantic Information Interoperability in Smart Spaces. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*, Cambride, UK, 22-25 November 2009, ACM, 2009; pp. 158–159.
58. Dierks, T. and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. 2008. Available online: http://www.ietf.org/rfc/rfc5246.txt (accessed on 23 November 2012).
59. Suomalainen, J. Flexible Security Deployment in Smart Spaces. In *Proceedings of the International Workshops, S3E, HWTS, Doctoral Colloquium, Held in Conjunction with GPC 2011*, Oulu, Finland, 11–13 May 2011, Springer: Berlin-Heidelberg, Germany, 2012; pp. 34–43.

Series title and number
VTT Science 50

Title	**Adaptive security in smart spaces**
Author(s)	Antti Evesti
Abstract	Smart spaces – like smart homes, smart offices and smart cities – exploit various resources in order to offer enriched services and information for the end users. Achieving security in such a dynamic and heterogeneous environment with predefined and static security mechanisms is a challenging task. Hence, solutions for self-adaptive security are needed. Self-adaptive security is able to automatically select security mechanisms and their parameters at runtime in order to preserve the required security level in a changing environment. The research problem of the dissertation is how to achieve security adaptation in a smart-space application. For this dissertation, architecture and knowledge base objectives were set. The objectives were satisfied with security-adaptation architecture that contains an adaptation loop and an ontology-based knowledge base for security. The adaptation loop conforms to the Monitor, Analyse, Plan, Execute and Knowledge (MAPE-K) model, which is a widely applied reference model in autonomic computing. The ontology-based knowledge base offers input knowledge for security adaptation. The research was carried using five cases, which iteratively developed the architecture and the knowledge base for security adaptation. The contributions of the dissertation are: Firstly, reusable adaptation architecture for security is presented. The architecture strictly conforms to the MAPE-K reference model and defines all phases in it. Moreover, the architecture is the first that specifically separates security knowledge from the adaptation loop. Secondly, the architecture supports the utilisation of security measures to recognise an adaptation need. Security measures are presented by means of a three-level structure in order to achieve systematic monitoring. Due to the suggested architecture, it is possible to reuse and extend the defined security measures. Thirdly, this is the first time that an ontology has been applied for security adaptation. Hence, the Information Security Measuring Ontology (ISMO) acts as the knowledge base for the security adaptation. The ISMO is applicable at design-time and runtime alike. At design-time, the ISMO offers knowledge for the software architect, in order to design an application with security-adaptation features. In contrast, the application searches knowledge from the ISMO at runtime, in order to automatically perform the security adaptation. Utilising the ontology as a knowledge base ensures that the knowledge is presented in a reusable and extensible form. Moreover, the application does not need hard-coded adaptation knowledge.
ISBN, ISSN	ISBN 978-951-38-8113-9 (soft back ed.) ISSN 2242-119X (soft back ed.) ISBN 978-951-38-8114-6 (URL: http://www.vtt.fi/publications/index.jsp) ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)
Date	December 2013
Language	English, Finnish abstract
Pages	71 p. + app. 119 p.
Name of the project	
Commissioned by	
Keywords	Architecture, security measuring, ontology, knowledge base, self-adaptive
Publisher	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland, Tel. 020 722 111

Julkaisun sarja ja numero
VTT Science 50

Nimeke	**Älytilojen mukautuva tietoturva**
Tekijä(t)	Antti Evesti
Tiivistelmä	Älytilat, kuten älykodit, älytoimistot ja älykaupungit, hyödyntävät monenlaisia resursseja tarjotakseen loppukäyttäjille parempia palveluita ja informaatiota. Muuttuvissa ja heterogeenisissa älytiloissa on haastavaa saavuttaa tietoturvaa ennalta määrätyillä ja muuttumattomilla tietoturvaratkaisuilla. Tämän vuoksi tarvitaan mukautuvaa tietoturvaa. Mukautuvassa tietoturvassa tietoturvamekanismit ja -parametrit valitaan automaattisesti suorituksen aikana, jotta vaadittu tietoturvataso saavutetaan myös muuttuvassa älytilassa. Väitöskirjan tutkimusongelmana on, kuinka saavutetaan mukautuva tietoturva älytilan sovelluksessa. Tutkimusongelma on jaettu arkkitehtuuri- ja tietämyskantatavoitteiksi. Tavoitteet täytetään mukautuvan tietoturvan arkkitehtuurilla, joka sisältää mukauttamissilmukan ja ontologiapohjaisen tietämyskannan tietoturvalle. Mukauttamissilmukka noudattaa MAPE-K-mallia, Monitoroi (M), Analysoi (A), Suunnittele (P), Toteuta (E) ja Tietämys (K), joka on yleisesti käytetty referenssimalli autonomisissa ohjelmistoissa. Väitöskirjassa MAPE-K-mallin tietämysosa toteutetaan hyödyntäen ontologiapohjaista tietämyskantaa. Tutkimus on toteutettu käyttäen viittä tapaustutkimusta, joissa kehitetään mukautuvan tietoturvan arkkitehtuuria ja tietämyskantaa iteratiivisesti. Väitöskirjan tulokset ovat: i) Työ esittää uudelleenkäytettävän mukauttamisarkkitehtuurin tietoturvalle noudattaen tarkasti MAPE-K-referenssimallia ja määritellen mallin kaikki vaiheet. Lisäksi arkkitehtuuri on ensimmäinen, joka täsmällisesti erottaa tietoturvatietämyksen mukauttamissilmukasta. ii) Arkkitehtuuri tukee mukauttamistarpeen havaitsemista tietoturvamittareiden avulla. Tietoturvamittarit esitetään kolmetasoisena rakenteena, joka mahdollistaa systemaattisen monitoroinnin. Lisäksi kehitetty arkkitehtuuri mahdollistaa tietoturvamittareiden uudelleenkäytön ja laajentamisen. iii) Työssä sovelletaan ensimmäistä kertaa ontologiaa tietoturvan mukauttamiseen. Tietoturvan mittausontologia (ISMO) toimii tietämyskantana mukautuvalle tietoturvalle. ISMO soveltuu sekä suunnitteluaikaiseen että suorituksenaikaiseen käyttöön. Suunnitteluaikana ISMO tarjoaa ohjelmistoarkkitehdille tietoa mukautuvan tietoturvan toteuttamiseksi osaksi sovellusta. Suorituksen aikana sovellus puolestaan etsii tietoa ISMO:sta suorittaakseen mukauttamisen. Ontologian käyttäminen tietämyskantana mahdollistaa tiedon uudelleenkäytön ja laajennettavuuden. Lisäksi sovelluksessa ei tarvita kovakoodattua tietämystä tietoturvan mukauttamiseen.
ISBN, ISSN	ISBN 978-951-38-8113-9 (nid.) ISSN 2242-119X (nid.) ISBN 978-951-38-8114-6 (URL: http://www.vtt.fi/publications/index.jsp) ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)
Julkaisuaika	Joulukuu 2013
Kieli	Englanti, suomenkielinen tiivistelmä
Sivumäärä	71 s. + liitt. 119 s.
Projektin nimi	
Toimeksiantajat	
Avainsanat	Architecture, security measuring, ontology, knowledge base, self-adaptive
Julkaisija	VTT PL 1000, 02044 VTT, Puh. 020 722 111

VTT SCIENCE 50 Adaptive security in smart spaces

ISBN 978-951-38-8113-9 (Soft back ed.)
ISBN 978-951-38-8114-6 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN-L 2242-119X
ISSN 2242-119X (Print)
ISSN 2242-1203 (Online)