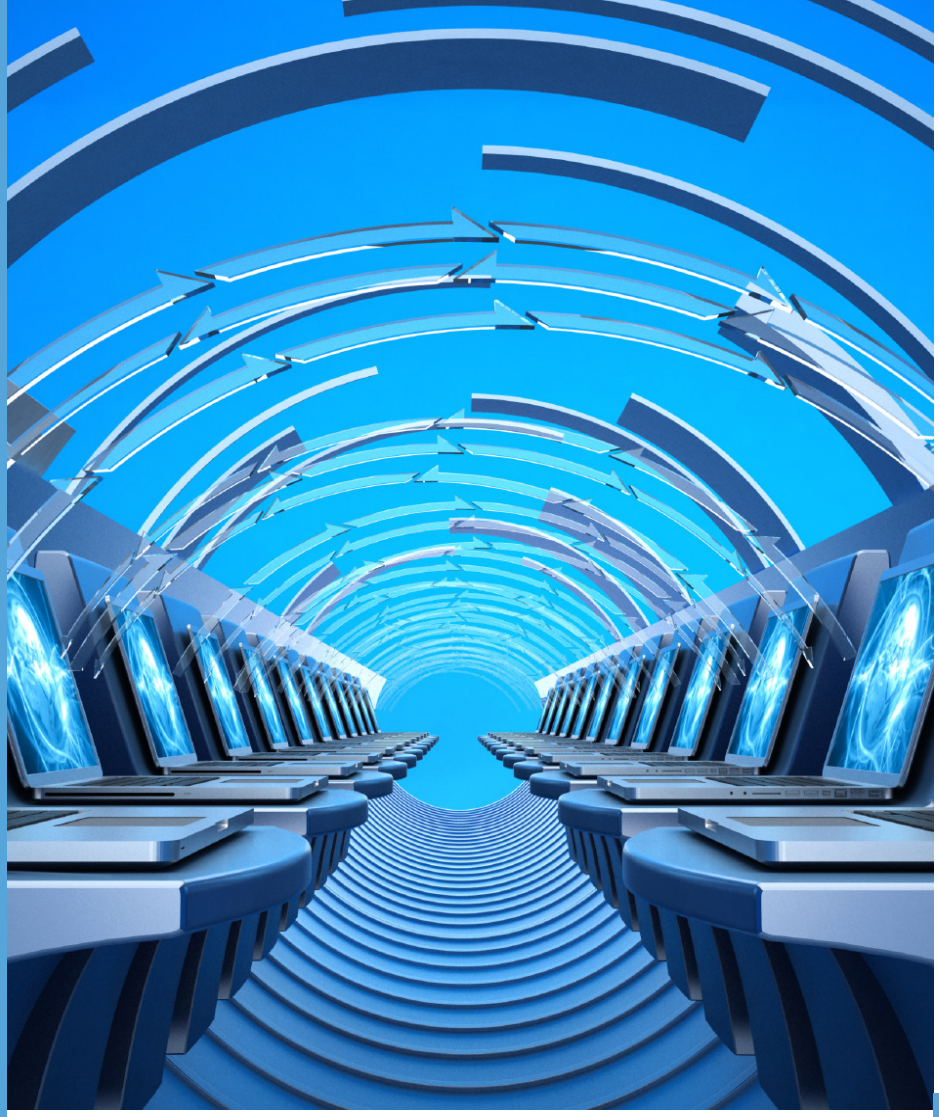

SCIENCE • TECHNOLOGY • RESEARCH HIGHLIGHTS • VISIONS

Dissertation
37

Reusable, semantic, and context-aware micro-architecture

Approach to managing interoperability and dynamics in smart spaces

Susanna Pantsar-Syväniemi

VTT

VTT SCIENCE 37

Reusable, semantic, and context-aware micro-architecture

Approach to managing interoperability and dynamics in smart spaces

Susanna Pantsar-Syväniemi

Thesis for the degree of Doctor of Science in Technology, to be presented with due permission for public examination and criticism in OP-sali (Auditorium L10), at the University of Oulu, on the 28th of August, 2013, at 12 noon.

ISBN 978-951-38-8009-5 (Soft back ed.)
ISBN 978-951-38-8010-1 (URL: http://www.vtt.fi/publications/index.jsp)

VTT Science 37

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

Reusable, semantic, and context-aware micro-architecture
Approach to managing interoperability and dynamics in smart spaces

Uudelleenkäytettävä, semanttinen ja kontekstitietoinen pienoisarkkitehtuuri. Menetelmä yhteentoimivuuden ja dynamiikan hallintaan älykkäissä tiloissa. **Susanna Pantsar-Syväniemi.** Espoo 2013. VTT Science 37. 72 p. + app. 122 p.

Abstract

The amount of shared information has increased a great deal in ubiquitous systems, where the previously isolated devices and appliances have become part of the system and are producing or consuming the information. The ubiquitous system, or the smart environment, lacks an approach that supports scalability and enables semantic interoperability. It is challenging to provide a dynamic behavior at the run time without human intervention. A number of dedicated solutions have been developed for the ubiquitous environment because of its complexity. The dedicated solutions are usually non reusable.

An approach is needed that i) is reusable as such or partly, ii) provides the semantic interoperability, iii) enables dynamic and behavioral interoperability between the receiver and sender of the information at run time, and iv) is scalable by being modular, and decoupled.

This thesis proposes a novel approach to managing interoperability and dynamics in smart spaces. The approach includes a Context-Aware Micro-Architecture (CAMA), and a Context Ontology for Smart Spaces (CO4SS). This approach is independent of implementation languages and communication techniques. CAMA, as an architectural pattern, is usable without its semantic support, CO4SS. In the literature, it is the first approach that fulfills the requirements that are set for a context data distribution system.

The power in CAMA relies on the usage of the standard and web-based techniques, in the separation-of-concerns principle, and in the enhanced control loop, MAPE-K. The latter has four parts, Monitor; Analyze; Plan; Execute that share Knowledge. CAMA is highly dynamic, which is due to the run-time updatable rules. The creation of the rules is laborious, as they are written into text boxes of Message Sequence Charts. This will be improved when new tools are developed for the rule creation. Additional research is needed to validate the scalability of the approach with a “Big data”. CO4SS can be widened with the domain-specific and quality ontologies. It supports the evolution management of the smart space: all smart spaces and their applications ‘understand’ the common language that is defined by it. CO4SS has the potential to be a de facto ontology for the context-aware, i.e., intelligent applications.

Key words ontology, software architecture, embedded, ubiquitous system, design pattern

Uudelleenkäytettävä, semanttinen ja kontekstitietoinen pienoisarkkitehtuuri
Menetelmä yhteentoimivuuden ja dynamiikan hallintaan älykkäissä tiloissa

Reusable, semantic, and context-aware micro-architecture. Approach to managing interoperability and dynamics in smart spaces. **Susanna Pantsar-Syväniemi.**
Espoo 2013. VTT Science 37. 72 s. + liitt. 122 s.

Tiivistelmä

Kaikkialla läsnä oleva, ns. Ubi-järjestelmä, sisältää paljon yhteistä tietoa, jonka määrä kasvaa, kun ennen erillään toimineet laitteet tulevat osaksi järjestelmää. Ubi-järjestelmä, tai älykäs tila, tarvitsee uusia menetelmiä, jotka tukevat järjestelmän skaalautuvuutta sekä mahdollistavat semanttisen yhteentoimivuuden eri laitteiden ja applikaatioiden välillä. Tällainen järjestelmä on dynaaminen, ja on haasteellista saada sitä tukevaa automaattista toimintaa toimimaan reaaliajassa. Ubi-järjestelmä on myös monimutkainen, ja siksi olemassa olevat järjestelmät ovat olleet hyvin erikoistuneita, eivät uudelleenkäytettäviä.

Tarvitaan siis ratkaisumalli, joka i) on uudelleenkäytettävä sellaisenaan tai osittain, ii) tarjoaa semanttisen yhteentoimivuuden, iii) mahdollistaa dynaamisen ja yhteentoimivan toiminnan reaaliajassa tiedon lähettäjän ja vastaanottajan välillä ja iv) on skaalautuva. Väitöstyössä on kehitetty uusi menetelmä yhteentoimivuuden ja dynamiikan hallintaan älykkäissä tiloissa. Menetelmässä on kontekstitietoinen mikroarkkitehtuuri (CAMA) sekä kontekstiontologia (CO4SS). Ne ovat riippumattomia toteutuskielistä ja kommunikointiteknologioista. CAMA on arkkitehtuurimalli, ja se on käytettävissä ilman semanttista tukeaan eli kontekstiontologiaa, CO4SS. Menetelmä on kirjallisuudessa ensimmäinen, joka täyttää kontekstitiedon jakelujärjestelmän vaatimukset.

Mikroarkkitehtuurin vahvuus on siinä, että se käyttää sekä standardoituja että web-pohjaisia tekniikoita. Arkkitehtuuri pitää kontekstinhallinnan erillään muusta ohjelmistosta ja hyödyntää parannettua MAPE-K mallia, jonka neljä osaa – monitorointi (M), analysointi (A), suunnittelu (P) ja toteutus (E) – hyödyntävät tietämystä (K). Reaaliaikaisesti päivitettävät säännöt tekevät kehitetystä mikroarkkitehtuurista hyvin dynaamisen. Sääntöjen luonti on työlästä, koska säännöt kuvataan tekstilaatikkoina toimintajärjestyskuviin. Tämä paranee tulevaisuudessa, kunhan uusia työkaluja sääntöjen kuvaamiseen saadaan kehitettyä. Menetelmän skaalautuvuutta suurien datamäärien kanssa on vielä tarpeen tutkia lisää. Kontekstiontologia on laajennettavissa niin sovellusalue- kuin laatuontologioilla. Se tukee älytilojen evoluutiota tarjoamalla yleisen kielen, jota kaikki älytilat ja niihin liittyvät ohjelmistot ymmärtävät. Ontologialla on edellytyksiä kehittyä ns. de facto -kontekstiontologiaksi, kun luodaan kontekstitietoisia eli älykkäitä ohjelmistoja.

Avainsanat ontology, software architecture, embedded, ubiquitous system, design pattern

Preface

The research work that is reported in this thesis was carried out in the Service Architectures and Systems group at VTT Technical Research Centre of Finland and in the Advanced Research Center on Electronic Systems for Information and Communication Technologies E. De Castro (ARCES) at the University of Bologna, Italy. The majority of research work was carried out during the years 2009–2012 in the Artemis Smart Objects For Intelligent Applications (SOFIA) project.

I am grateful to Research Professor Eila Ovaska for her guidance and support. She encouraged me to carry out part of the work abroad. She asked me whether I could imagine working in Bologna when we were at the airport of Bologna on the way back to Oulu from one of the first meetings of SOFIA project. As is known, I responded positively. I wish to express my gratitude to Professor Tullio Salmon Cinotti for allowing me to work in his group in the ARCES, guiding my work, and helping my family in many ways during our one year stay in Bologna, Italy.

I am also grateful to Professor Olli Silvén for supervising my work and bringing me through the last steps. I wish to thank Professor Tommi Mikkonen and Doctor Davy Preuveneers for reviewing the manuscript of this thesis. Their feedback was valuable and it helped me a lot to sharpen the presentation of thesis. I am indebted to Research Professor Aarne Mämmelä for his advice during my thesis work and, especially, while I was writing the thesis. I wish to thank Mr Seppo Keränen for checking the language.

I wish to give my sincere thanks to all my co-workers during this work: Paolo Bellavista, Antti Evesti, Susanna Ferrari, Jarkko Kuusijärvi, Sandra Mattarozzi, Valerio Nannini, Eila Ovaska, Anu Purhonen, Luca Roffia, Tullio Salmon Cinotti, Kirsti Simula, Jorma Taramaa, and Guido Zamagni.

Publication II was the first paper for me as the main author and without help and support of Jorma Tarmaa and Eila Ovaska (former Niemelä) it might never have been published. Ned Chapin, as an editor for Journal of Software Maintenance and Evolution, was patiently waiting for the author's maternity leave to end before kindly pushing to receive the revised version.

This thesis is financially supported by the VTT, the Finnish Funding Agency of Technology and Innovation (Tekes), the European Union, the Seppo Säynäjäkangas Science Foundation, the Emil Aaltonen Foundation, and the KAUTE Foundation.

I would like to thank my family of their patience and support during this work. My husband Mika has made it possible to concentrate on my research in many evenings and weekends, without forgetting the year in Bologna. My children, Atte, Aino, and Anna, have been a powerful counterpart to my research.

Don't join an easy crowd;
you won't grow. Go where
the expectations and
the demands are high.
-Jim Rohn

Kempele, June 2013 Susanna Pantsar-Syväniemi

Academic dissertation

Supervisor
Olli Silvén
University of Oulu
Faculty of Technology, Department of Computer Science and Engineering,
P.O. Box 4500, FI-90014 University of Oulu, Finland

Advisor
Eila Ovaska
VTT Technical Research Centre of Finland
Digital Services Research
P.O. Box 1100, FI-90571 Oulu, Finland

Reviewers
Tommi Mikkonen
Tampere University of Technology
Department of Pervasive Computing
P.O. BOX 553, FI-33101 Tampere, Finland

Davy Preuveneers
KU Leuven
Department of Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Opponent
Jan Bosch
Chalmers University of Technology
Department of Computer Science and Engineering
Maskingränd 2, SE-41258 Gothenburg, Sweden

List of publications

This thesis is based on the following original publications, which are referred to in the text as I–IX and reprinted in Appendices. The publications are reproduced with kind permission from the publishers.

I Pantsar-Syväniemi, S. (2012) Architecting Embedded Software for Context-Aware Systems. In: *Embedded System – Theory and Design Methodology.* Tanaka, K. (ed.). InTech, ISBN 979-953-307-580-7, Rijeka, Croatia, 123–142.

II Pantsar-Syväniemi, S., Taramaa, J., Niemelä, E. (2006) Organizational evolution of digital signal processing software development. *Journal of Software Maintenance and Evolution: Research and Practice* **18**(4): 293–305.

III Pantsar-Syväniemi, S., Purhonen, A., Ovaska, E., Kuusijärvi, J., Evesti, A. (2012) Situation-Based and Self-Adaptive Applications for Smart Environment. *Journal of Ambient Intelligence and Smart Environments* **4**(6): 491–516.

IV Toninelli, A., Pantsar-Syväniemi, S., Bellavista, P., Ovaska, E. (2009) Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability. *Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing (M-MPAC 2009)*, ACM; Article No. 5.

V Pantsar-Syväniemi, S., Simula, K., Ovaska, E. (2010) Context-awareness in Smart Spaces. *Proceedings of the First International Workshop on Semantic Interoperability for Smart Space (SISS 2010)*, IEEE, 1023–1028.

VI Pantsar-Syväniemi, S., Kuusijärvi, J., Ovaska, E. (2011) Context-awareness Micro-architecture for Smart Spaces. *The 6th International Conference on Grid and Pervasive Computing (GPC 2011),* Springer-Verlag Berlin Heidelberg LNCS 6646/2011, 148–157.

VII Pantsar-Syväniemi, S., Kuusijärvi, J., Ovaska, E. (2012) Supporting situation-awareness in Smart Spaces. *Proceedings of the First International Workshop on Self-managing Solutions for Smart Environments (S3E 2011) organized in association with the GPC 2011 6th International Conference on Grid and Pervasive Computing (GPC 2011),* Springer-Verlag Berlin Heidelberg, LNCS 7096/2012, 14–23.

VIII Evesti, A., Pantsar-Syväniemi, S. (2010) Towards Micro Architecture for Security Adaptation. *Proceedings of the 4th European Conference on Software Architecture. ECSA 2010 workshops: 1st International Workshop on Measurability of Security in Software Architectures (MeSSA 2010),* ACM, 181–188.

IX Pantsar-Syväniemi, S., Ovaska, E., Ferrari, S., Salmon Cinotti, T., Zamagni, G., Roffia, L., Mattarozzi, S., Nannini, V. (2011) Case Study: Context-aware Supervision of a Smart Maintenance Process. *The 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT2011)*, *The Second International Workshop on Semantic Interoperability for Smart Space (SISS 2011)*, IEEE Computer Society, 309–314.

Author’s contributions

The writing of Publication I is done solely and Publication II mainly by the author. These publications are affected by experiences gathered during embedded software development at Nokia Siemens Networks (formerly Nokia Networks and Nokia Telecommunications). The writing of Publications V, VI, VII, and IX is mainly carried out by the author, who was responsible for the research carried out in these publications. In Publication III, the author contributed to the adaptation framework on behalf of the context-awareness agents, the CO4SS, the ontology mapping and being the main author. The author has co-operated with Eila Ovaska, Alessandra Toninelli, and Paolo Bellavista for the research and writing of Publication IV. In Publication VIII, the author contributed to the most relevant context for information security and participated in the writing process. The research ideas are validated by the implementation work carried out by Jarkko Kuusijärvi in Publications III, VI, VII and Susanna Ferrari in Publication IX.

Contents

Publications IV, VI, VII, VIII are not included in the PDF version. Please order the printed version to get the complete publication (http://www.vtt.fi/publications/index.jsp).

List of abbreviations

2G	2nd generation mobile network. GSM being a major standard
3G	3rd generation mobile network. UMTS being a major standard
4G	4th generation mobile network. LTE being a major standard
ANSI	American National Standards Institute
ARCES	Advanced Research Center on Electronic Systems for Information and Communication Technologies E. De Castro, University of Bologna, Italy
ARTEMIS	European Technology Platform in the field of embedded systems
BTS	Base Transceiver Station
CAMA	Context-Aware Micro-Architecture
CAMPUS	Context-Aware Middleware for Pervasive and Ubiquitous Service
CO4SS	Context Ontology for Smart Spaces
CoDAMoS	Context-Driven Adaptation of Mobile Services
CONON	CONtext ONtology
CoOL	Context Ontology Language
DSP	Digital Signal Processor
DSPL	Dynamic SPL
FOAF	Friend Of A Friend ontology
GSM	Global System for Mobile Communications, Groupe Spécial Mobile (in the beginning)
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IOP	InterOperability Platform

ISMO	Information Security Measuring Ontology
ISO	International Organization for Standards
KAUTE	Kaupallisten ja teknillisten tieteiden tukisäätiö
KP	Knowledge Processor
KPI	Knowledge Processor Interface
LTE	Long Term Evolution
M3	Multipart, Multidevice, and Multivendor framework
MAPE-K	Control loop of self-adaptation, decomposed into monitoring (M), analyzing (A), planning (P) and execution (E). K means knowledge
MDA	Model Driven Architecture, OMG's standard
MSC	Message Sequence Chart
OMG	Object Management Group
OWL	Web Ontology Language, W3C's recommendation
OSGi	Open Services Gateway Initiative, a set of specifications that defines a dynamic component system for Java. OSGi is developed by OSGi alliance
QoC	Quality of Context
QoS	Quality of Service
RDF	Resource Description Framework, W3C's recommendation
RF	Radio Frequency
RIBS	RDF Information Base Solution, a SIB for resource limited devices
RPM	Run-time Performance Management ontology
SDR	Software-Defined Radio
SeMaPS	Self-Management Pervasive Service ontology
SIB	Semantic Information Broker
Smart-M3	implementation based on the IOP
SOAP	Simple Object Access Protocol, XML-based, W3C's specification
SOFIA	Smart Objects For Intelligent Applications, the European Artemis programme under the subprogramme "Smart environments and scalable digital service"
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SPARQL	Simple Protocol and RDF Query Language, W3C's standard

SPL	Software Product Line
SQL	Structured Query Language, a standard of the ANSI, and of the ISO
SSAP	Smart Space Application Protocol, XML based interaction protocol
SWRL	Semantic Web Rule Language, W3C's proposal
Tekes	the Finnish Funding Agency of Technology and Innovation
UML	Unified Modeling Language, OMG's specification
UMTS	Universal Mobile Telecommunication System
VTT	VTT Technical Research Centre of Finland
W3C	WWW Consortium
Web	Short for World Wide Web
Wi-Fi	Wi-Fi Alliance, certifies interoperability of WLAN products based on IEEE 802.11 standards
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
WWW	World Wide Web
XML	eXtensible Markup Language, W3C's specification

1. Introduction

To offer situation-based services for human beings, even proactively, is a great challenge, especially for software. In addition to the software, computing devices and a network are needed. These three technologies – i) cheap, low-power computers, ii) software for ubiquitous applications, and iii) a network that ties everything together – were mentioned in Mark Weiser's vision [1] over two decades ago, when he envisioned the disappearance of technologies that weave themselves into the fabric of everyday life, until they are indistinguishable from it. This ubiquitous computing vision included the invisibility aim, but assumed the solutions to appear later that will work out the technical challenges related to scalability, interoperability and dynamics. This thesis delves deeper into software technologies, by presenting reusable, semantic, and context-aware software agent-based micro-architecture for managing the semantic, dynamic and behavioral interoperability in ubiquitous systems.

Ten years after Weiser's vision, in 2001, Mahadev Satyanarayanan [2] discussed the challenges in the field of pervasive computing and emphasized that ubiquitous computing could also be called pervasive computing. He introduced how pervasive computing i) relates to distributed systems and mobile computing, and ii) is affected by the effective use of smart spaces, invisibility, localized scalability, and techniques for masking the uneven conditioning of environments. He concluded that research has challenges, e.g., i) in the area on software agents, and ii) expert systems and artificial intelligence. The former relates to high-level proactive behavior and the latter has to do with decision making and planning.

In 2003, Debashis Saha and Amitava Mukherjee [3] presented pervasive computing as a paradigm for the 21st century. Added to devices, networking, and applications, the authors highlighted middleware to be the fourth necessity for the pervasive computing. Figure 1 presents their system view of pervasive computing. This extension is achieved by integrating pervasiveness support technologies, such as interoperability, scalability, smartness, and invisibility. The smartness is attached to a context management that involves accurate sensing followed by intelligent control or action between machines and humans. The authors also state that perception, or context-awareness, is an intrinsic characteristic of intelligent environments. Although these support technologies describe the challenges of the

pervasive or ubiquitous system and they were introduced in more detail in [3] than in Weiser's vision, the solutions for the challenges were still missing.

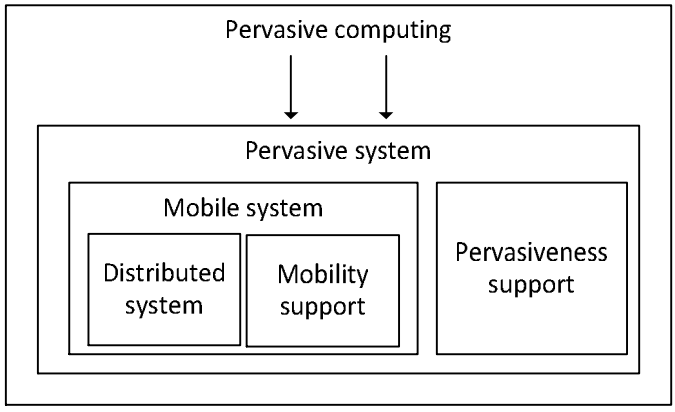

Figure 1. System view of pervasive computing [3].

The cooperation in the environment of ubiquitous computing is based on the information. This calls for interoperability solutions provided by intelligent applications. Interoperability can be abstracted to be a sharing of information between a sender and a receiver and it can be divided into levels, as shown in Figure 2 and presented in Publication III. This thesis provides an approach to reaching semantic, dynamic, and behavioral interoperability.

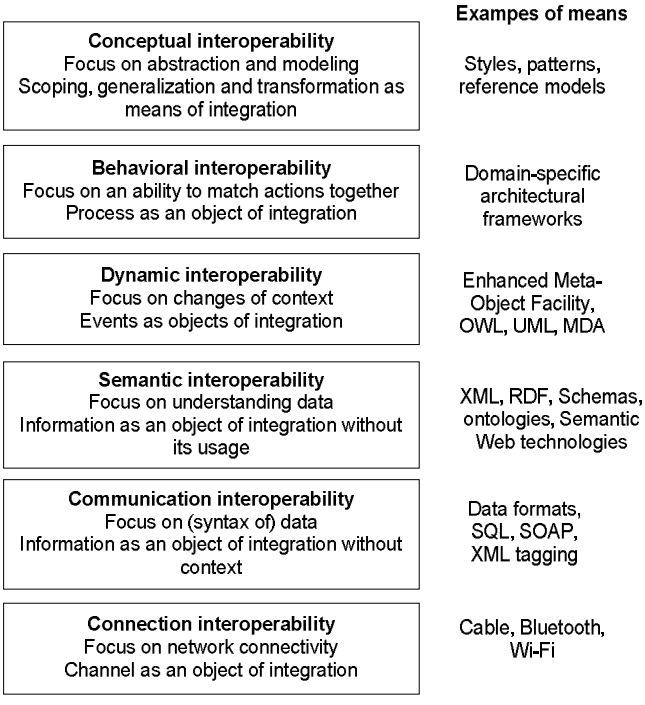

Figure 2. Interoperability levels of the smart environment [Publication III].

1.1 Information – the basis for the cooperation

Charles W. Morris, in 1938, presented that science and signs are inseparately interconnected and that a sign refers to something for someone [4]. He wrote that the process in which something functions as a sign may be called semiosis. He also presented three dimensions of semiosis:

1. A formal relation of signs to one another is called a syntactical dimension of semiosis, *syntactics*.
2. A semantic dimension of semiosis is called *semantics*, which refers to the relations of signs to the objects to which the signs are applicable.
3. The relation of the signs to interpreters is called the pragmatical dimension of semiosis, *pragmatics*.

He wrote the foundations of the theory of signs, the semiotics, which could – in principle, be presented as a deductive system. The work has widely been considered a step towards unification of science. He seems to have succeeded in that as the dimensions stated above have also been used in software engineering. However, the technologies needed for building a deductive system were ignored. The

dimensions can be thought to represent levels, so that syntactics is the first level that correlates with the communication interoperability level of Figure 2. Semantics is the second level and correlates with the semantic interoperability level, and pragmatics is the third level correlating with the dynamic interoperability level.

Warren Weaver, in 1949, presented a theory to consider the information via three levels from 1 to 3 that respectively relates to technical, semantic, and efficiency problems [5]. The technical problems were, e.g., concerned with the accuracy of transference from the sender to the receiver of sets of symbols, or of a continuously varying signal, or of a continuously varying two-dimensional pattern. The semantical problems were concerned with the identity, or satisfactorily close approximation, in the interpretation of meaning by a receiver, as compared with the intended meaning of a sender. The effectiveness problems were concerned with the success with which the meaning conveyed to the receiver leads to the desired conduct on his part. Weaver's categorization is quite similar to Morris's, while he seemed to have developed his approach independently of the work done by Morris.

Claude E. Shannon, in his information theory, in 1948, stated that the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. He presented that the messages have a *meaning* that they refer to or are correlated according to some system with certain physical or conceptual entities [5]. He stated that these semantic aspects of communication were irrelevant to the engineering problem. Thus, his work concentrated on the first level of Weaver's theory.

Later on, in the 1980's, Werner Gitt stated that the concept of information had become as fundamental and far-reaching as energy and matter were [6]. Therefore, he presented that the information was the third fundamental quantity. He declared that although Shannon's concept of information was adequate for dealing with the storage and transmission of data, it would fail when trying to understand the qualitative nature of information. Gitt introduced five levels of information: statistics, syntactics, semantics, pragmatics, and apobetics. His levels correlate with those of Morris and Weaver, but he has divided their first and third levels. Gitt correlated his own work to the work of Shannon, even though Shannon ignored the semantics and pragmatics levels of the information. Figure 3 presents the levels of information by Gitt.

The cooperation in the intelligent environment is based on the information as mentioned above. The information layers from one to three relate directly to the three lowest levels of the interoperability categorization, as is presented in Figure 2: i) the statistics to the connection interoperability, ii) the syntactics to the communication interoperability, and iii) the semantics to the semantical interoperability. The fourth layer, the pragmatics, relates to both the dynamic and behavioral interoperability levels. The fifth layer, the apobetics, relates to the reaching of the goal by the information. Apobetics has no counterpart in the interoperability levels, even though the desired goal is attainable by semantic, dynamic, and behavioral interoperability.

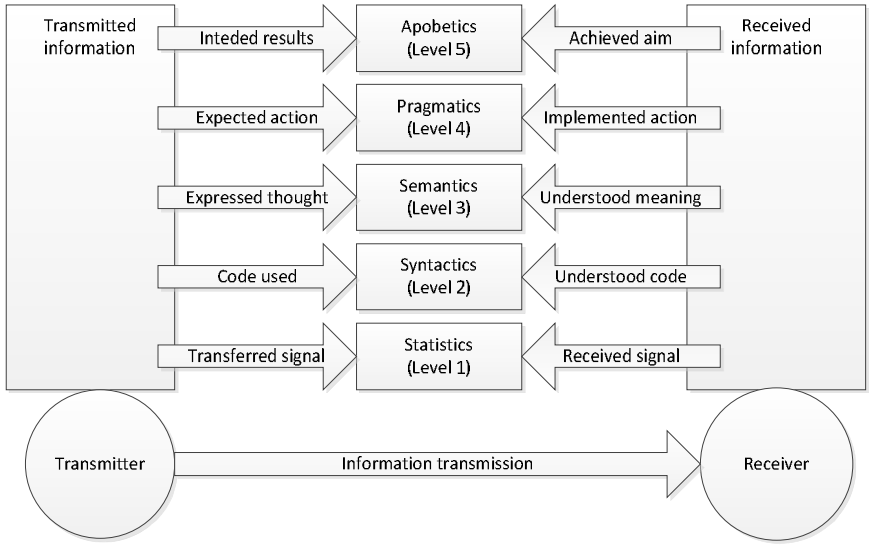

Figure 3. The levels of information [6].

Figure 3 presents a generalized view of information sharing between one transmitter and one receiver. The levels of information remain, even though the number of transmitters or receivers increase. The scaling of transmitters and receivers is usually designed in advance and managed with standards and interface specifications in the embedded system, such as a base station (BTS) that is a part of the mobile network infrastructure. The amount of shared information increases a great deal in the ubiquitous system, as the previously isolated devices and appliances will be part of the system and are producing or consuming the information. The ubiquitous system, or the smart environment, is still lacking an approach that would support scalability and enable interoperability. Invisibility has not been in focus in the information theory. The challenges for the applications are residing in the third and fourth layers: semantics and pragmatics. Thus, the approach presented in this thesis needs to offer the semantics and techniques for the pragmatics, to comply with the dynamic and behavioral features of the ubiquitous system.

The aim of ubiquitous computing is to enable a smart environment, where physical and digital environments will be integrated and interoperable. The cooperation in the intelligent environment is based on the information gathered, stored, transmitted, transformed, inferred, and anticipated. The Internet is the de facto network for data transmission and the Semantic Web provides the formats and language to integrate and combine the data that is originated from heterogeneous sources and to relate the data to the sources, i.e., to real world things.

1.2 Software for intelligent applications

Ubiquitous environment is seen, in [7], to lend itself to autonomic computing, because of the complexity of installing and maintaining such an environment and keeping it running in a stable way. One of the challenges in autonomic computing is the requirement to carry out robust software engineering, not only to provide solidly built autonomic systems, but systems capable of interoperating with each other [7]. The interoperability and the complexity are the challenges facing intelligent applications. It is also challenging to provide a dynamic behavior at the run time without human intervention.

Interoperability is usually provided via specifications issued by standardization organizations or alliances. A good example of a successful standardization effort is the 2G mobile network and its successors, 3G and 4G. The set of specifications has covered the overall network, e.g., a communication between a mobile device and BTS. The communication is based on a fixed radio resource allocation. A certain portion of the radio spectrum is assigned or licensed on a long term basis and covering large regions, such as entire countries [8]. This approach provides ultimate protection from harmful interference in an allocated radio band, but the fixed radio frequency allocation may lead towards significant underutilization of the radio spectrum, due to a highly sporadic usage across different geographical regions, as well as in different periods of time [8]. Therefore, a cognitive radio is viewed as an approach for improving the use of the radio spectrum [9].

The cognitive radio, built on a software-defined radio (SDR), is an intelligent wireless communication system that is aware of its surrounding environment, and uses the methodology of understanding-by-building to learn from the environment and to adapt its internal states to statistical variations in the incoming radio frequency (RF) stimuli by making corresponding changes in certain operating parameters in real time, with two primary objectives in mind: i) highly reliable communication whenever and wherever it is needed, and ii) efficient use of the radio spectrum [9]. The intelligence for the cognitive radio is created, in [10], by using a cognition cycle. This cycle is time-sensitive and immediately acts when it is needed or according to a plan-decide-act cycle. The learning path is also included and can be used while not performing time-sensitive acts. The cognitive radio employs software agents that are driven by knowledge. The work that is reported in this thesis uses a similar approach. Another similarity is the usage of micro-“things”: the cognitive radio consists of multiple micro-worlds [10] and this work is proposing to approach the system development via micro-architectures.

The ubiquitous environment does not have the same kind of overall system architecture as mobile networks do. This shortage has delayed the arrival of a ubiquitous ecosystem. In the ubiquitous environment, the interoperability is achievable by a context model. Different models have been introduced for the context design [11]: key-value, markup scheme, graphical, object-oriented, logic-based, and ontology-based. The ontological models of context are emphasized to provide clear advantages for heterogeneity and interoperability [12].

Context-awareness has been emphasized to be a key factor for ubiquitous applications [13]. The complexity of the ubiquitous environment has led to emergence of various dedicated solutions. These include a blackboard-based software framework and a tool for mobile device context awareness [14], a formal verification method for situation definitions [15], a resource optimized quality assured context mediation framework [16], different approaches for a scalable tracking of mobile object and an efficient processing on continuous spatial range queries [17], a context data management system [18], and a Context-Aware Middleware for Pervasive and Ubiquitous Service (CAMPUS) [19]. These dedicated solutions are not as reusable as is the approach of this thesis.

The context data distribution system, envisioned by Bellavista et al. in [20], is presented via a context data delivery layer, context data management layer, and run-time adaptation support, as illustrated in Figure 4. The work of this thesis, the context-aware micro-architecture with the software agents and the context ontology corresponds to the context data distribution functionality and precisely to the context data management and run-time adaptation support. This relation is illustrated later in Figure 10 in Section 3.1.

The logical architecture of the context data distribution system also includes the context data sources and sinks. The context data management includes, e.g., the context data security, which is problematic from the modularity and decoupling point of view. The approach of this thesis is more modular, as it does not have an embedded management of the security or other quality attributes to its functionality. Run-time security management [21] and the run-time performance management [22] are divided into separate micro-architectures. An adaptation framework, presented in Publication III, gathers together the adaptation functionalities of security, performance, and context management to ease the trade-off decisions in the ubiquitous system.

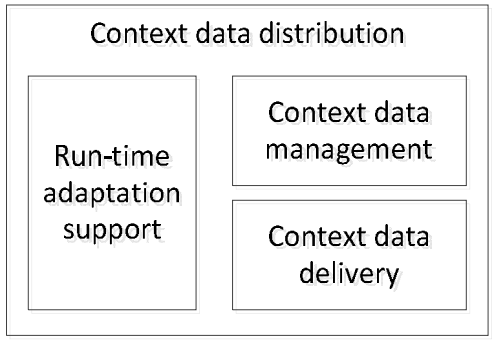

Figure 4. The context data distribution [20].

The requirements for context data distribution [20] are introduced next. The first requirement relates to context data delivery and falls, partly, outside the scope of this thesis. The other requirements are relevant for this work. These requirements

are used for validating the approach of this thesis, as described in Sections 3. and 3.1. Bellavista et al. do not emphasize interoperability in their survey and the authors have not taken into account newer middleware or approaches like Hydra [23], CAMPUS [19] [24], MUSIC [25], or an IOP developed in the SOFIA-project [26]. The scalability of the systems or middleware is also omitted in their survey.

i) Bellavista et al. highlight that the context data production and consumption should be possible at different times, and sinks and sources do not have to know each other. Thus, both time and space decoupling has to exist.

ii) The second requirement stresses that only currently required context data needs to be distributed, and the distribution has to adapt to available resources, such as computational capabilities, and wireless standards.

iii) The third requirement emphasizes that context data typically has a limited visibility scope that depends on physical or logical locality.

iv) The fourth requirement is about an enforcement of QoC-based constraints, such as data delivery time and reliability, to enable correct system management.

v) The fifth requirement relates to handling the life cycle of context data and highlights that the context data distribution should be able to self-control the data distribution process.

1.3 Problem statement

The complexity of ubiquitous computing has promoted dedicated context-aware approaches and some of these are presented in Section 1.2. These approaches are not reusable or have not been able to adapt the behavior of the ubiquitous system at run time. A recent survey [20] related to context-aware middleware solutions shows that the existing context-aware middleware solutions do not fulfill the requirements set for the context data delivery approach. The survey has embedded context data security for context data management, which is problematic from the modularity and decoupling point of view.
Therefore, there is a need for an approach that

1. is reusable as such or partly,
2. provides semantic interoperability,
3. enables dynamic and behavioral interoperability between the receiver and sender of the information at run time,
4. is scalable by being modular, and decoupled.

Ubiquitous systems are dynamic by nature, as there are frequent changes in the amount of users and devices within them. Therefore, the design methods, architectural patterns and principles need to be studied if they are to be revised.

1.4 Goals and scope

Thus, the goal of this research is to develop an approach that supports i) interoperability by the semantics of information, and ii) scalability by being modular, and decoupled. The approach is also supposed to perform context management at run time by adapting its behavior according to the situations and supporting other software agents to adapt accordingly. The context management is to be tunable at run time and also reusable as such or partly. To fulfill the goal of interoperability, the relevant context ontologies need to be surveyed and used as much as possible. This is due to the need to generate a context model that forms the interoperability basement by being generic and expandable with domain specific concepts. The context model should not include concepts that are application specific or needed for measuring or handling qualities. The software requires semantics to be interoperable and intelligent. This thesis will focus on the ontological context model as this model is likely to have potential for achieving interoperability between different and heterogeneous devices and their embedded smart software.

The approach we are looking for needs to be scalable to be able to provide context management from personal to home and office spaces toward city environment. The scalability asks for a relatively simple approach that would be easy to understand, natural, and not complicated to use beside of being modular and decoupled. The approach will support invisibility by being nondisturbing and embedded as much as possible. The simplicity helps to manage the complexity of the ubiquitous system. The complexity can be simplified by, e.g., dividing it according to the characteristics or functionality of the system. The simplification can also be done by changing from distributed to centralized computing. This is the aim in distributed baseband computing used in the cloud as a centralized pool for meeting the constantly changing capacity and coverage demands on the mobile network. These demands arise as the users of smartphones, tablets, and other smart devices switch applications and devices at different times and places [27].

The complexity of ubiquitous or autonomic computing was manifested by Paul Horn, from IBM. The author represents the perspective of managing the complexity by concentrating on the characteristics of the computing system [28]. This manifesto has been inspired by the human autonomic nervous systems that are adaptive by nature. The IBM manifesto presented eight characteristics for autonomic applications or systems, which were later formulated by Parashar and Hariri, in [29], as follows:

1. *Self Awareness:* An autonomic application/system “knows itself” and is aware of its state and its behavior.
2. *Self Configuring*: An autonomic application/system should be able to configure and reconfigure itself under varying and unpredictable conditions.
3. *Self Optimizing*: An autonomic application/system should be able to detect suboptimal behavior and optimize itself to improve its execution.

4. *Self-Healing*: An autonomic application/system should be able to detect potenteial problems, recover from problems, and continue to function smoothly.
5. *Self Protecting*: An autonomic application/system should be capable of detecting and protecting its resources from both internal and external attacks and maintaining the overall system security and integrity.
6. *Context Aware*: An autonomic application/system should be aware of its execution environment and able to react to changes in the environment.
7. *Open*: An autonomic application/system should be able to function in a heterogeneous world and should be portable across multiple hardware and software architectures. Consequently, it must be built on standard and open protocols and interfaces.
8. *Anticipatory:* An autonomic application/system should be able to anticipate, to the greatest possible extent, its needs and behavior and those of its context, and be able to manage itself proactively.

Based on the experience of a huge, embedded, and real-time system, the author believes that the manifesto has relevant but high-level requirements for designing systems like the BTS of a mobile network. The manifesto is open to various interpretations, as it describes the autonomic system in an abstract and mildly idealistic way, while not providing any technological solutions for building the system. Based on the experience, the author concentrates on the first and the sixth requirement of the manifesto: self-awareness and context-awareness. Context-awareness in this thesis includes both of the awareness requirements. This focus is due to the experience that the system first has to know itself and its environment to be able to fulfill the second, the third, the fourth, the fifth, and the eighth requirement – to configure itself, to optimize itself, to recover, to protect itself and to be proactive. The seventh requirement, to be open, is important for reaching interoperability. It guides the selection of development techniques or removes restrictions where the cognitive radio is aiming at by reusing the available radio spectrum.

The context data distribution system [20] does not use the IBM manifesto by any means, even though there are several research results [7] [29] [30] [31] [32] [33], in addition to the work reported in this thesis, that use the manifesto and a closed-loop control as a basis for their approaches, surveys or systems. Bellavista et al. in their new survey [20] stress that the context data distribution is still missing a viable approach for the self-adaptation of data processing, i.e., aggregation and filtering, at run time. This thesis fulfills this need except for the context delivery part. This is due to the available technical solutions for the context data delivery.

The novel and distilled results of this thesis are:

1. A context management approach that supports i) interoperability by the semantics of information, and ii) scalability by being modular and decoupled. It performs at the run time by adapting its behavior according to the situation and by supporting other software agents to adapt accordingly. It is tunable at the run time and also reusable as such or in part.

2. An ontological context model for enabling and managing semantic interoperability. It is generic and expandable for heterogeneous domains and runtime quality management.
3. Demonstrations that validate the developed approach and the ontological context model.
4. A proposal to exploit the approach and the context ontology also in areas other than smart spaces. The usage possibilities are estimated in the digital BTS based on the experience gathered, in particular.
5. Knowledge in the area of information theory, autonomic computing, dynamic software product lines, and embedded systems.
6. A proposal to revise SPL for the development of dynamic and context-aware systems.

1.5 Research approach and method

The contributions of this thesis are based on the experiences gathered during embedded software development at Nokia Networks from 1993 to 2008 and, subsequently, on research at VTT Technical Research Centre of Finland. The industrial experience of the huge embedded software system at Nokia Networks forms a sound basis for building up a new approach to developing context-aware applications. Figure 5 illustrates how this research has proceeded, from software development practices to the development of applications.

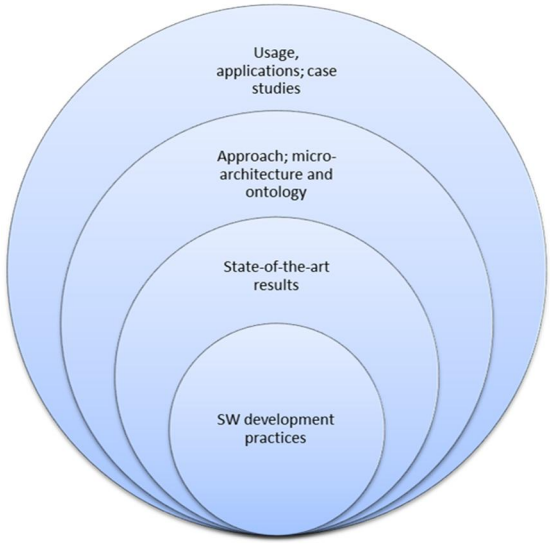

Figure 5. The path of this research, from grass-root practices to the development of applications.

The work began by studying the state-of-the-art results of software product line (SPL), which is a well-known development practice to enhance the productivity of the software development organization. An organizational evolution of digital signal processing (DSP) software development was studied and compared with both the organizational models for the SPL and a multistage model for the software life cycle. A new organizational model for product line development was introduced. The new hybrid model clarifies long-term responsibilities in large software organizations with hundreds of staff members and formulates the organization according to the software architecture. The evolution phase in the multistage model was extended for BTS DSP software.

The work continued by studying context-awareness and then extending the interoperability platform of the SOFIA project [26] to support context-awareness. According to the separation-of-concerns principle [34] [35], context-awareness was proposed to be separated from the application logic to its own software agents. This was done to increase reusability and to decrease complexity. In addition, the research done on embedded software product development led to the insight that a reusable asset needs to be small and simple, not complex. With these criteria in mind, a conceptual model for supporting context-awareness was created. Thus, the foundation for context management in a ubiquitous system was created. This novel model included a context ontology and a concept for context-awareness that would enable context monitoring, context reasoning, and context-based adaptation.

The concept was further developed and introduced as a reusable approach to reach context-awareness. This new approach was called context-aware micro-architecture (CAMA). The micro-architecture is a set of patterns used together to realize parts of a system or subsystem [36]. CAMA provides a solution for managing an adaptation based on the context. It consists of three types of software agents: context monitoring, context reasoning, and context-based adaptation agents. These agents do not communicate directly with each other as the information flows via a semantic database. Each agent is usable as such, can be duplicated when needed, and is updatable at the run time. The first result is thus achieved.

The context ontology of the model was also developed further and published as a context ontology for smart spaces (CO4SS). CO4SS is generic and extensive with its six contextual dimensions: physical, digital, situational, user, social, and historical. Consequently, the second result is achieved.

This research was carried out within real and laboratory environments, and this is why case studies have been used as the main research method. The case studies involved either a single case or multiple cases, and numerous levels of analysis [37]. If the case is studied as a whole, it is called a holistic case study, whereas if the case includes multiple units of analysis, it is called an embedded case study. According to the guidelines introduced in [38], the case study is a suitable research method for software engineering research since it studies contemporary phenomena in their natural context. In software engineering, the case may be a software development project [39].

The feasibility of the developed approach, including CAMA and CO4SS, is demonstrated in four application development cases. The aim in the first application was to use CAMA to create the context-aware agents that were configured during the design time. This application was intended for activating the required functionality according to the rules and existing situations in the smart home, i.e., to 1) wake-up the user according to the first meeting of the current day (if it is a workday) and the time usually used for the morning activities after wake-up and 2) prepare coffee when the user is woken up. The aim in the second application was to use CO4SS for adding the necessary functionality at the development time and to check its ability to expand with domain-specific parts that were created to control the lighting. The second application was in the smart home to switch on the lighting at the right time, according to the preferences set by the user.

The third application had to do with a smart maintenance scenario. The aim was to check i) the mapping of CO4SS and domain-specific ontology, and ii) the reusability and reconfigurability of the context monitor agent at the run time. The context was used to supervise the progress of the maintenance scenario. The fourth application was GuideMe, with the aim of exploiting context, security, and performance information for adapting the service according to the quality requirements and the context of the user, as well the smart environment, without bothering the end-user. The implemented GuideMe was able to identify situations and adapt itself accordingly. The GuideMe i) enhanced navigation, e.g., according to the destination of the driver and available parking slots near by the destination and ii) rerouted if there was an accident on the selected route. The configuration was done at the run time. The third result is achieved with these successful applications.

The fourth result is presented in Section 5.5, which discusses the use in areas other than the smart space. The fifth result provides new knowledge, which was created by this research and presented in Sections 5.3 and 5.4. The sixth result relates to the proposal to revise the SPL, which is presented in Publication I. Table 1, Table 2, and Table 3 clarify the contributions of Publications I to IX, and the thesis according to the research steps.

Table 1. Contributions to the state of the art.

Research step	**Ref**	**Contribution**
Study state-of-the-art	II	Study existing organization models for SPL and a multistage model for software life cycle. Compare both models in view of the organizational evolution of DSP software development.
	III	Evaluate existing interoperability models, architectures for smart environment, and context-aware and self-adaptive solutions for networked devices.
	VI	Evaluate middleware platforms that i) provide a context storage and management layer, and ii) comply with the dimensions for decentralization, portability, and interoperability.
	VII	Evaluate context ontologies that i) provide convenient, general, and relevant concepts, and ii) are suitable for heterogeneous smart spaces.
	I	Evaluate the influencing factors when architecting embedded and real-time software for digital BTSs and compare their usefulness for context-aware systems. Propose to revise the SPL if needed.
	Thesis	Supplementary study in the area of context-aware middleware solutions, and the context ontologies.

Table 2. Contributions to the approach.

Approach	IV	Develop a context-awareness support for the interoperability platform (IOP). Follow the separation-of-concerns principle.
	V	Develop a concept for semantic agents for managing context.
	VI	Create context-aware agents that are updatable at run time and reusable.
	VII	Develop a generic context ontology for smart spaces which is expandable and can be aligned with separate ontologies. The generic context ontology should provide concepts, such as software agents, for providing services and rules for defining use situations. The ontology should not include concepts that are application specific or needed for measuring or handling qualities.

Table 3. Contributions to validating the researched and developed approach.

Application	IX	Test the context-based adaptation in the smart maintenance scenario, where the context and domain-specific ontologies are mapped for a context-aware supervision feature. The Context Monitor agent is reconfigurable at run time.
	VIII	Test the usage of context monitoring in run-time security management.
	III	Test the usage of CAMA and CO4SS, as they are merged to the adaptation framework and used in the navigation scenario. Run-time configuration is used.
	VI	Test CAMA with CO4SS to activate the required functionality, according to the rules and existing situations in the smart home. This is a cross-domain (personal space and the user's smart home) scenario to wake the owner up with a design-time configuration.
	VII	Test CO4SS and CAMA in the smart home to switch the lighting on at the right time according to the preferences set by the user. This is the cross-domain scenario between the personal space and the home of the user, where CO4SS is used at development time and it is expanded with domain specific parts that are created to control the lighting in the home.

1.6 Summary of the publications and overview of the thesis

This thesis consists of an introductory part and nine original publications. The introductory part presents the objectives and scope of this work and summarizes the contributions. Chapter 2 presents the background for the thesis. The reusable and semantic approach to develop context-aware applications is described in Chapter 3. The validation of the proposed solution is introduced in Chapter 4. Chapter 5 discusses the theoretical consequences, practical influence and the recommendations for further research. Chapter 6 concludes the thesis.

This thesis consists of nine publications. Publication I represents a review from the architecture perspective, when the system is either real-time and embedded in the industrial setting or ubiquitous in the research computing. The significance of the context is also addressed. To gain real advantage of or to be able to use the new methods for software development, the relevant hardware support needs to exist. The revised SPL approach is proposed to guide the architecture via an understanding of the eligible ecosystem toward small functionalities or subsystems. Each of these subsystems is a micro-architecture with a unique role.

Publication II studies existing organization models for SPL and a multistage model for the software life cycle. It compares both models in view of the organizational evolution of DSP software development. Publication II has been written mainly by the author. Jorma Taramaa pointed out the relevant references for the telecommunication domain and Eila Ovaska (former Niemelä) for the state of the art for software architectures.

Publication III introduces the adaptation framework for the situation-based and self-adaptive applications. It provides state of the art analyses of the existing interoperability models, architectures for the smart environment, and context-aware and self-adaptive solutions for networked devices. CAMA is embedded to the framework with CO4SS, which is updated by mapping it with a run-time performance management ontology (RPM). The adaptation framework is used to develop a GuideMe application and the development is successful. The development process is also illustrated. The author contributed to the adaptation framework concerning the context-awareness agents, CO4SS, ontology mapping and being the main author for the publication. Jarkko Kuusijärvi was responsible for the implementation work for Publication III.

Publication IV proposes an extension to the IOP to support context-awareness. The extension follows the separation-of-concerns principle. The concept of IOP and the primary challenges in the design of smart space applications are also illustrated. This publication argues that to achieve the greatest possible scalability in supporting context-awareness in smart environments, context-awareness needs to be detached from the semantic database into separate facilities.

Publication V presents a set of new capabilities for achieving context-awareness in smart spaces. The capabilities are a novel context ontology and a set of software agents for context monitoring, reasoning and context-based adaptation. The use of the context-awareness concept is exemplified by an emergency scenario of a smart city. The writing of Publication V was mainly carried out by the author of this dissertation. Kirsti Simula brought her knowledge of the software agents with beliefs, desires, and intents to the publication. She also created rules, based on the Semantic Web Rule Language (SWRL) to check the feasibility of the proposed context ontology. Due to space limits, these rules had to be left out of the publication.

Publication VI describes the context-aware micro-architecture for designing software as one that enables reacting to the current situation or being proactive and taking upcoming circumstances into account. The CAMA approach is also enhanced from the context-awareness concept presented in Publication II. The enhanced CAMA is used in two scenarios, which are instantiated to a personal smart space and a smart home. The writing of Publication VI was mainly carried out by the author. The implementation to validate the proposed work was carried out by Jarkko Kuusijärvi.

Publication VII presents the context ontology for smart spaces. CO4SS is created, based on the work published in Publications II and III. CO4SS is used to switch on lights according to the calculated wake-up time and user preference. The context monitoring agent has been a motivator for the creation of a reusable monitor [40] by the colleagues of the author. The writing of Publication VII was mainly carried out by the author of this dissertation. The implementation to validate the proposed work was carried out by Jarkko Kuusijärvi.

Publication VIII introduces the use of context monitoring in a micro-architecture for a security adaptation. The relevant context information that affects the information security in smart spaces is described. This approach is used, e.g., in a

greenhouse demonstrator. Publication VIII is co-authored with Antti Evesti. The author of this dissertation was responsible for the context ontologies and context monitoring, while also contributing to the most relevant context for information security.

Publication IX describes a case study where CAMA and CO4SS are used as a part of the smart maintenance process of a building. The Context Monitor agent is designed and implemented to supervise the phases of a building maintenance process. A new way for binding the context information relevant for the context monitoring agent is also presented. The writing of Publication IX was mainly carried out by the author of this dissertation. The implementation was done by Susanna Ferrari, as a part of her thesis work (Tesi di Laurea) at the University of Bologna, Italy.

2. Background

This chapter presents the relevant background information on software architecture, software development based on the reuse principle and software development for context-aware applications in embedded and ubiquitous systems. First, the software architecture and reuse are introduced and then relevant architectural patterns are presented. Context-awareness and situation-awareness are surveyed. Last, the context ontologies are presented.

2.1 Software architecture and reuse-based software development

Software architecture is an important element in each software system. It is used to illustrate the structure and the environment of a software system. Software architecture shares information about, e.g., the partitioning of the system to smaller parts, the deployment to the processors, and the priorities of the tasks. Publication I presents the most common definitions for software architecture. The role of the software architecture grew in BTS DSP software development during the 1990s and 2000s. This and other characteristics of DSP software development, such as the use of a processor-specific assembler until the end of the 1990s, is described in Publication II.

Software architecture has a key role when managing software development. The exploitation of previous knowledge or the outputs of software development is called software reuse. Software reuse can be defined as the process of creating software systems from existing software, rather than building software systems from scratch [41]. It has been emphasized to be a key method for significantly improving quality and productivity [42]. Software reuse requires the commonalities and variabilities to be found in the system under development. Publication II states that the commitment for software reuse needs to exist at each level of the software development organization and that software reuse needs to be planned and organized.

Software reuse based on SPLs has been successfully applied in the software industry and several cases have been reported in various publications [43] [44] [45] [46] [47] [48]. The SPL engineering is about finding the commonalities and variabilities. SPL can be defined as follows:

- “A software product line consists of product line architecture and a set of reusable components that are designed for incorporation into the product line architecture.” [45]
- “Beyond simple reuse or a component-based development strategy, a software product line lets an organization manage and evolve its product family holistically, as a single, unified entity.” [49]
- “A software product line is a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of reusable core assets in a prescribed way.” [50]

The organizational evolution of DSP software development is compared with the organizational models for SPL. The results of the comparison analyses as well the organizational models are presented in more detail in Publication II.

SPL is a valid approach when seeking the benefits in modifying the products of a product group. The main process-centric SPL approaches are surveyed in Publication I. Based on this survey and the development experience gathered in the industry, the SPL approach is recommended to be revised to better suit context-aware systems. The revision is needed for advising the software architect to concentrate on small functionalities or subsystems after catching the overall understanding of the whole context-aware system. These subsystems are micro-architectures, having a dedicated role like performance, energy-efficiency and security management or a more generic role like context management.

The micro-architectures are the building blocks for dynamic ubiquitous systems, where smart units cooperate with each other and with human beings. The micro-architectures are to be designed from the bottom-up instead of top-down, as is the case with the SPL approach. As stated in Publication I, the micro-architecture should be configurable at design time, at instantiation time and during run-time.

SPLs are evolved to dynamic SPLs (DSPL), which extend existing SPL approaches by moving their capabilities to the run time [51]. DSPLs are considered to be comparable with the other approaches for developing adaptive systems [51]. DSPLs do not have any support for interoperability. Their support for the system changes all the time is minimal or even non-existing, because of the coming and going of devices, appliances or user. DSPLs guide the application to be built as a monolith and from the top-down. Therefore, the revised SPL is more suitable than DSPL to be used for architecting context-aware systems.

2.2 Architectural patterns and principles

As mentioned earlier, the aim of this thesis is to find an approach that is reusable, simple, semantic and that also offers context management at the run time. According to the reuse and the separation-of-concerns principle, we are looking for an approach that is a micro-architecture and supports context-awareness. The separation-of-concerns principle means that context-awareness will be separated from

the remainder of the software system. This also complies with the reuse principle, as context-awareness is a commonality for the ubiquitous as well as for embedded systems.

To manage something within software, such as the context, the thing to be managed needs to be monitored and analyzed. Usually the monitored data and the results of the analysis are saved for future reference. Therefore, an architectural pattern: a MAPE-K loop [52] [30], is exploited in the developed solution to manage the context. IBM introduced the MAPE-K loop as a new approach for a feedback control mechanism to monitor the behavior of the system and to enable taking appropriate actions [52]. The MAPE-K loop, presented in Figure 6, consists of four phases: Monitor, Analyze, Plan, and Execute, all of which share Knowledge. The loop is called an autonomic manager. The four parts are working together to provide a control loop functionality and they are supposed to collaborate using asynchronous communication techniques, such as a messaging bus.

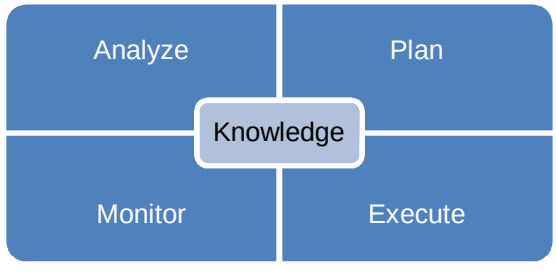

Figure 6. MAPE-K loop.

The MAPE-K loop has been used for communicating the architectural aspects of autonomic systems that are also called self-managing systems or self-adaptive systems [7]. The MAPE-K loop has also been applied in the MUSIC middleware [25] and in a conceptual model of DSPL [53]. As mentioned earlier in this thesis, autonomic computing is suitable for ubiquitous computing to enable intelligent behavior. This research uses the MAPE-K loop in a new manner, by using a database to share the knowledge. The MAPE-K loop has been stated [7] to be a software component and that the knowledge is shared inside the loop. The approach of this thesis brings more granularity to the MAPE-K loop i) by using a semantic database to share the knowledge instead of, e.g., sending messages inside the MAPE-K loop, and ii) by decoupling the MAPE parts to separate agents in order to promote scalability. The semantic database forms the basis for interoperability. It is used “globally” in the system and not only locally by the proposed approach.

The multipart, multidevice, and multivendor (M3) is a baseline architecture with three interoperability levels – device, service, and information. The IOP, developed in the SOFIA-project [26], complies with the M3 concept and supports information interoperability by means of Semantic Information Brokers (SIB). The SIB forms a backbone, which stores all the information as Resource Description Framework

(RDF) triples. The triples are *subject-predicate-object* expressions, where the *subject* presents the resource to be described, the *predicate* the type of a property relevant to this resource, while the *object* can be data or another resource. Thus, the SIB constitutes an RDF database and the RDF [54] is a directed, labeled graph data format for representing information on the Web.

The software agents can connect to the SIB and exchange information through a Smart Space Access Protocol (SSAP) that is an eXtesible Markup Language (XML) based interaction protocol. The software agent is called a Knowledge Processor (KP) in the IOP. The agents use a Knowledge Processor Interface (KPI) to communicate with the SIB. The KPI provides SSAP operations to join, leave, insert, remove, update, query, subscribe, and unsubscribe. Three different instantiations of the SIB are created: The Smart-M3 SIB for resource rich devices and systems without real-time requirements [55] [56], the SOFIA Application Development Kit (ADK) for simulation purposes with Java, and the RDF Information Broker Service (RIBS) [57] for resource constraint devices with high security and performance requirements.

The work that is reported in this dissertation will combine the IOP and the MAPE-K loop in a single solution and will also extend the IOP with a context-awareness capability. The proposed solution, the micro-architecture for managing the context, requires an RDF-based database, but not necessary any of the introduced IOP-based SIBs. Figure 7 presents the communication via an RDB-based database between different micro-architectures. Publication III presents how the interoperability levels were elaborated, cf. Figure 2, in order to better adapt them to the development of smart environments and their applications. Basically, the first three levels (from bottom to top) are relatively similar to the levels of the C4IF [58] and M3 models [26].

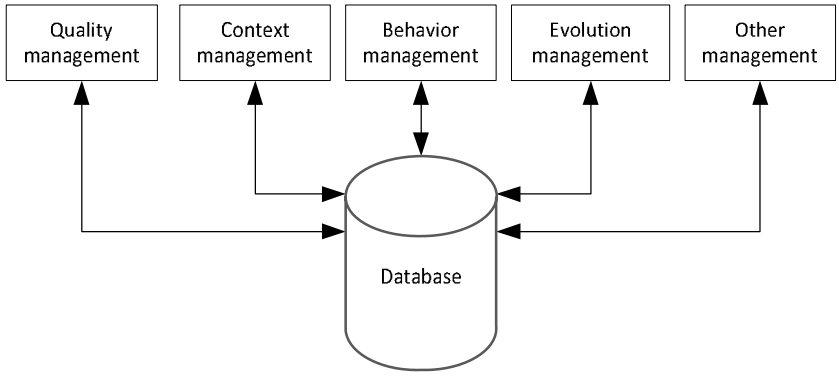

Figure 7. Micro-architectures communicate via an RDF-based database.

2.3 Context-awareness and situation-awareness

It is necessary to introduce the context as a part of the software system architecture. In an embedded software system, the external context of a subsystem is drawn to show the dependencies with the other subsystems. The internal context is also illustrated to show the dependencies between the different functionalities or design parts inside the subsystem. The context of embedded software system is discussed in Publication I. In ubiquitous systems, the context is often drawn via the user and his/her location. There has been a great deal of discussion about the context: what it is, how to model it, and how to use it to implement a ubiquitous application. The most cited definitions for context and context-awareness have been given by Dey and Abowd [59]:

- "*Context* is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered to be relevant to the interaction between a user and an application, including the user and the application themselves".
- "*Context-awareness* is a property of a system that uses context to provide relevant information or services to the user, where the relevancy depends on the user's task".

The importance of context-awareness has been pointed out in various research results on pervasive and ubiquitous computing [12] [13] [60] [61] [62] [63]. Context-awareness will improve the software's ability to adapt to dynamic changes that are influenced by various factors during the operation of the software.

Context-awareness can be enhanced by *situation-awareness* that represents the ability to detect and reason about real-life situations [15]. Adaptations in context-aware applications are caused by a change in the situation, i.e., a change of a context value triggers an adaptation if the context update changes the situation [12]. The *situation* can be a simple one, an abstract state of a certain entity (e.g., a room is occupied), or a human action taking place in an environment (e.g., working or cooking) [64]. There are different kinds of relationships between situations: generalization, composition, dependence, contradiction, and temporal sequence [64]. Therefore, the handling of situations can be considered the most important feature to be supported by the micro-architecture of this dissertation. In embedded software, this kind of handling has usually been managed by state machines, whereas in ubiquitous software situations can be managed, e.g., by rules.

Embedded software is highly complex and its behavioral testing is challenging as the amount of states is huge. A recently published approach [65] proposes context-aware model-checking to be used for testing the behavior. The approach refers to use cases, or scenarios, as the context. The context describes how the environment interacts with the system. It corresponds to an operational phase identified as a system initialization or a reconfiguration. The context is illustrated by activity and sequence diagrams. The use of sequence diagrams is a common way to illustrate the behavior between the different actors. This work will also use

sequence diagrams to illustrate the flow of context information between the different software agents and the SIBs when creating applications to test the proposed solution. In an embedded software system, the environment is often well known, which is not the case in a ubiquitous software system. Ubiquitous software uses semantics, the ontological approach, to manage the heterogeneous environment.

2.4 Ontological context model

In connection with context and context-awareness, the role of ontologies has been emphasized [13] [60] [61] [66]. Ontological models of context are highlighted to provide clear advantages in terms of heterogeneity and interoperability [12], while there are also other approaches to modeling contexts as introduced by Strang and Linnhoff-Popien [11]. The approaches for context modeling are introduced in more detail in Publication I.

An *ontology* is an explicit specification of a conceptualization [67]. The ontology is also defined to be a shared knowledge standard or a knowledge model defining primitive concepts, relations, rules, and their instances [68]. As mentioned above in the Introduction, there is a need for new software approaches to enable semantic interoperability. Therefore, the ontological approach is used in this research to empower the software to understand the data, such as the value of the temperature, from heterogeneous sources.

An extensive work [69] compares the ontology-based models that are used in pervasive computing systems and provides details of the Context Ontology Language (CoOL) [70], the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [71], and the CONtext ONtology (CONON) [72] [73]. It also introduces an ontology for the Context-Driven Adaptation of Mobile Services (CoDAMoS) [74]. The work points out that SOUPA is the most consistent set of ontologies since it imports many of its concepts from external, consensual domain ontologies, such as the Friend-Of-A-Friend (FOAF) ontology. For more details on FOAF, see [75].

The work in [69] criticizes the ontology-based systems, due to their attempt to provide a single system-wide description rather than building complexity through composition. It concludes that i) composition is clearly important within an open pervasive system, and ii) individual issues, such as security and privacy, should be specified separately because they have an overall effect on the information that is collected, represented, exchanged and used. The author shares this conclusion and emphasizes the goal of this thesis, which is to develop a generic context ontology that is expandable and can be aligned with separate ontologies. The generic context ontology should employ concepts, such as software agents, for providing services and rules for defining different situations. It should not include concepts that are application specific or needed for measuring or handling qualities.

The generic context ontology should be composed of different dimensions, to enable it to be reusable, as such or partly. Being generic and reusable, it lays the foundation for the application designers to extend it with other ontologies required

by the applications at hand. The existing ontologies do not meet the requirements set for a generic ontology, even though a couple of contextual ontologies have been presented since the comparison work was published.

The ontologies published afterward are: i) the Context-Aware Middleware for Pervasive and Ubiquitous Service (CAMPUS) [19] [24], and ii) the Self-Management Pervasive Service (SeMaPS) [23]. CAMPUS was first introduced as CAMPO ontologies with CAMPUS middleware [24] and, at that time, it was divided into context, component, and application ontologies. In its new form, CAMPUS includes context, tasklet, and service ontologies [19]. It has mixed the concept of physical quantity to its context ontology. It does not use any concepts of situation. CAMPUS is dedicated to work with the CAMPUS middleware and it is targeted to enabling the changing of composition at run time. SeMaPS was presented with the Hydra Middleware, which is dependent on OSGi [76]. SeMaPS provides a device oriented set of ontologies for recognizing malfunctions and managing repairing actions as far as it is possible. The concept of situation is not presented. SeMaPS is a composition of many ontologies, including, e.g., the Quality of Service (QoS) ontology. SeMaPS and CAMPUS based systems support dynamic context, but not dynamically changing rules.

The IBM manifesto [28] and the autonomic computing architecture [52] do not recognize the ontological approach as a means to create an autonomic computing system. This is due to the fact that the ontological research has been activated only after the manifesto and it has begun to catch the research community's attention at the same time with the autonomic computing architecture. Seven years after the manifesto, in 2008, semantics is mentioned by Huebscher and McCann [7] as a challenge for the autonomic computing community. As a short summary, generic context ontology

- is suitable for all kinds of smart space applications;
- is reusable as such or in part, i.e., it is a composition of smaller ontologies;
- supports situations and rules;
- provides the semantics, the third level of information; cf. Figure 3; and
- enables semantic interoperability, cf. Figure 2.

3. Reusable, semantic, and context-aware approach

In this chapter, the developed approach is presented from the concept level to the context-aware micro-architecture. In addition, the creation of context ontology is introduced. The use of the micro-architecture and context ontology to develop situation-based and self-adaptive, i.e., context-aware, applications is described in Chapter 4.

The Interoperability Platform (IOP) combines the semantic representation of knowledge with an information-based interoperability model. The IOP is inspired by the shared semantic-based memory presented in [77], which does not specifically address the issue of device or technology heterogeneity in ubiquitous environments. A semantic distributed repository is proposed in [78] for the discovery and composition of information processing services. The semantic technologies are used, as in the IOP, to achieve information interoperability between data providers and consumers. The IOP supports any smart space application when the proposal in [78] focuses on service input or output modeling for workflow composition.

Two approaches to extending the IOP are identified and originally described in Publication IV. These approaches will be shortly introduced in the following as the related work forms the basis for the incremental development by the author of the reusable, semantic, and context-aware micro-architecture. The first approach is to extend the SIB and it is presented in Figure 8. The second approach is to offer context processing services as support software, as is shown in Figure 9. The SIB offers an interface for the applications and context-aware agents as illustrated by the form of a lollipop. The arrows represent the dependencies between the SIB and the applications or context facilities.

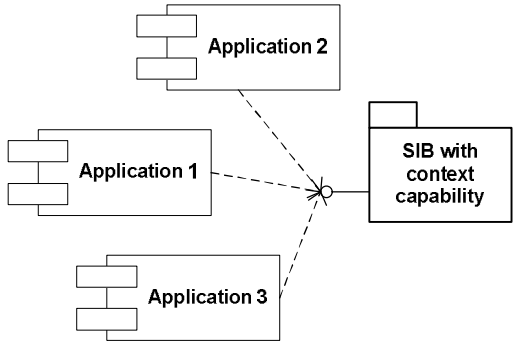

Figure 8. The extended SIB approach.

The extended SIB approach violates the separation-of-concerns principle, as the context management is embedded to the SIB. This approach does not scale well because the SIB is also burdened with the context-based computing. The support software approach fulfills the separation-of-concerns principle as the context processing is separated from the SIB and the applications. It also increases reusability as the context management can be used as such or in part or it can be not used, depending on the needs of the application developers. The support software approach seemed to be the better choice for providing context management.

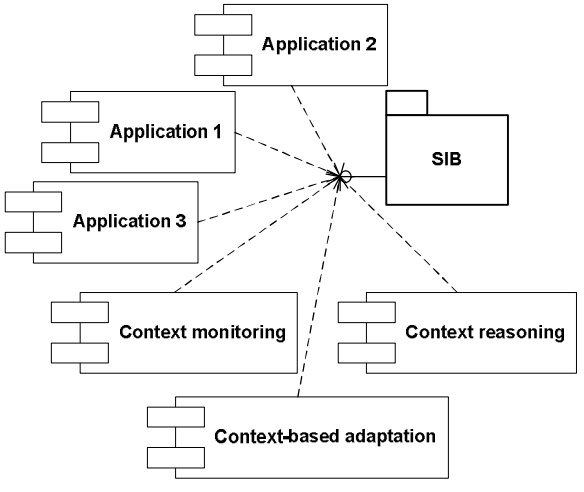

Figure 9. Context processing services as support software.

The support software approach complies with the first requirement set for the context data distribution system [20]: “The context data production and consumption should be possible at different times, and sinks and sources do not have to

know each other. Thus, both time and space decoupling has to exist." In addition to the separation-of-concerns principle, the support software approach follows the IOP's publish-subscribe communication between the SIB, the context facilities and the application agents. Therefore, data production and consumption is possible, asynchronously and anonymously. This approach is also modular as the context processing services are detached into separate agents. The support software approach supports i) scalability by being modular and decoupled, ii) interoperability with the semantics, and iii) invisibility by being nondisturbing and embedded as much as possible. The support software approach laid the foundation for the context management approach, i.e., the first result of the thesis.

3.1 Reusability

Publication II introduces the belief that it is vitally important to formulate the organization according to the software architecture, and it is essential to have a dedicated development organization with long-term responsibility for the software. Based on the experience, it can be stated, that without long-term responsibility, there is no software reuse. The natural way to decrease the software development efforts is to increase reuse. To successfully reuse software, the commitment to reuse has to exist at every level of the organization. Communication is most essential for getting this commitment. Software architecture is really important in sharing the information and the reasons; architecture is vital for reuse. The aim of creating a foundation for software reuse can quite easily be sacrificed because of business pressure or unclear responsibility for sharing between the line organization and project(s).

The strength of the software product line (SPL) is that it clarifies responsibility issues in creating, modifying and maintaining the software needed for the company's products. Publication I has identified that one key issue in the context-aware systems is to reuse the existing, e.g., communication technologies and devices, as much as possible, at least at the start of development, to minimize the amount of new things. Based on the experience and an experiment, it can be stated that a MARTE profile has been developed from a hardware design point of view because software reuse seems to have been neglected.

3.2 Context-awareness concept

Context management is a support layer functionality for intelligent applications and, therefore, it needs to be generic. As mentioned earlier in Chapter 2, context management uses a semantic database, such as the SIB, to share the knowledge. This increases modularity and reusability as each element in the context management communicates via the SIB and not directly with each other. The A and P of the MAPE-K loop were not feasible for the context management because of its generic nature. Those parts are meaningful, e.g., in the quality management as the run-time security management [21] or the run-time performance management [22].

The context management does not need the MAPE-K loop components Analyze (A) or Plan (P), as it is a servant for the other parts in the smart application. It needs a reasoning part instead. The author prefers to use adaptation as a term instead of execution in the context-awareness concept. Consequently, the concept needs to include elements for monitoring, reasoning and adaptation. These elements are software agents and they are presented in more detail in Section 3.2 and, originally, in Publications V and VI. Their usage is introduced in Chapter 4 and in Publications III, VI, VII, VIII, and IX.

In addition to the approach of this thesis, there are other approaches that have simplified the MAPE-K loop to three [7] [30] [31] [32] or even two [29] stages. The names of those stages vary according to the approach and they are introduced in the survey on self-healing systems [33]. The self-healing property is the 4th characteristic in the IBM manifesto. The approach of this thesis uses a database for the data-flow and, therefore, this approach does not have tight dependencies between stages as the other approaches do due to the fact that their data flows from stage to stage until the adaptation happens. The context-awareness concept was originally presented in Publication V.

The semantic knowledge increases the reusability and enables information interoperability. It is composed of a set of ontologies, where the context ontology plays the core role. The context ontology is expandable with relevant ontologies based on the needs of the application. The context ontology takes into account:

1. The varying resources: the used communication techniques and the computing capabilities. This data is continuously changing.
2. The user preferences and needs, also at run time.
3. The other users and possible users as the usage and the behavior of the smart application depends on the role of the smart space, which can be public, such as a smart city, or private, like a personal space.

The development of the context ontology was inspired by the SOUPA [71] ontology and the upper context conceptualization approach [61]. The former because it has been designed for smart spaces and it is agent-centric with beliefs, desires, and intents. The latter proposes that *User* is part of *Environment,* which is composed of *Digital Environment* and *Physical Environment*. The latter and the semantic context information triangle [12] are used as a basis for the creation of the context ontology. The triangle describes that the context information is layered from physical to digital and then to situational information. The first version of the context ontology is shown is Figure 11. The context ontology is presented in more detail in Section 3.3 and originally in Publication VII. Its use with context management agents is introduced in Chapter 4 and in Publications III, VI, VII, VIII, and IX.

The context-awareness concept relates to the pervasiveness support that is illustrated in Figure 1 and is described to constitute the pervasive middleware in [3]. The content of pervasive middleware is presented in Figure 4 and named as context data distribution functionality. The context-awareness concept, with the soft-

ware agents and the context ontology, corresponds to the context data distribution functionality as highlighted in Figure 10.

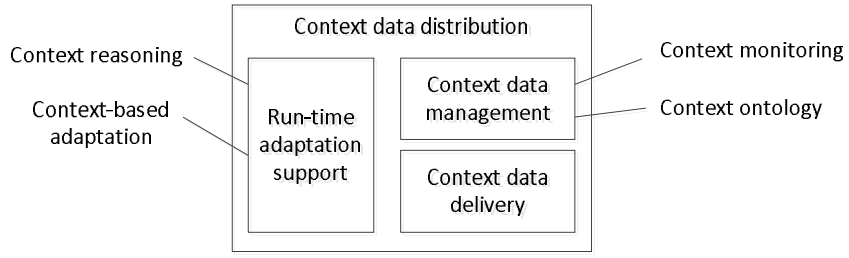

Figure 10. The context-awareness concept, in comparison with the context data distribution system described in [20].

The context-awareness concept fulfills the second requirement for the context data distribution system [20], which is: “Only currently required context data needs to be distributed, and the distribution has to adapt to available resources such as computational capabilities, and wireless standards.” It does this by offering the relevant context at the run time. The application also has its duties to configure the context management in such a way that only the relevant context is requested. The run-time performance management [22] takes care of, e.g., the computational capabilities.

The third requirement for the context data distribution system [20] is: “The context data typically has a limited visibility scope that depends on physical or logical locality.” This can be dealt with by appropriate reasoning based on the context. The reasoning is managed based on the rules that are updatable at run time, but that are, again, based on the needs of the application. The context-awareness concept takes care of the managing processes, while the application is responsible for the setting and updating of the rules.

The fourth requirement for the context data distribution system [20] is: “The context data distribution is about an enforcement of QoC-based constraints, such as the data delivery time and reliability, to enable the correct system management.” This requirement can be dealt with by the run-time quality management supported by the run-time context management. Publication III introduces the adaptation framework with the performance, security, and context management that will be enhanced by other quality attributes, such as reliability, in the future.

The fifth requirement for the context data distribution system [20] relates to handling the life cycle of context data and highlights that the context data distribution should be able to self-control the data distribution process. This can be fulfilled by the run-time context management, such as the context-awareness concept. The life cycle of context can be managed by an appropriate context model, such as the context ontology of this thesis, which is expandable by additional ontologies.

As far as the author is aware of the related work in the area of context data distribution systems, this work is the first one to fulfill the requirements set by Bellavista et al. in [20]. Their survey claims that there is a need for new approaches to solve the requirements set and they do not mention the work mainly done in the SOFIA-project [26], in which the IOP was extended with run-time context management. Other more recent studies were also omitted, such as Hydra [23], CAMPUS [19] [24], MUSIC [25]. This work differs from the aforementioned studies regarding at least three points: i) the use of the ontologies, ii) the management of the adaptation, and iii) the dependence on OSGi. Section 2.4 presents the reasons for CO4SS being more advanced than the ontologies related to Hydra and CAMPUS, and Section 5.2 compares MUSIC and Hydra with the work of this thesis. The MUSIC and Hydra middleware are presented in more detail in Publication III.

The context-awareness concept is reusable on many levels. It supports scalability because it is decoupled and modular. It uses a semantic database and uses an ontology for the context. Thus, it is also semantic and provides interoperability. It constitutes an embedded feature for providing run-time context management for the smart environment. Therefore, it also supports invisibility.

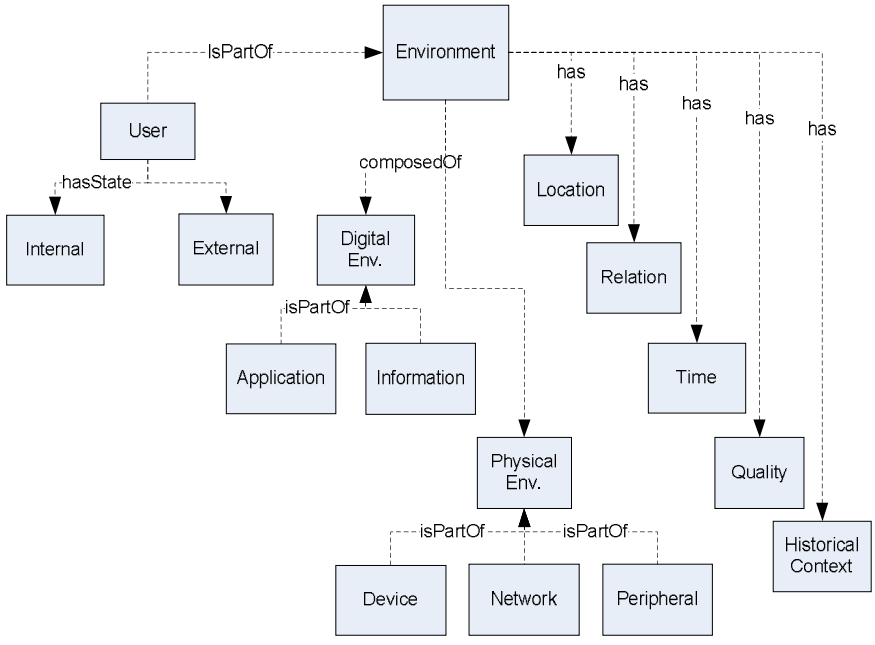

Figure 11. A simplified version of the context ontology.

3.3 Context-Aware Micro-Architecture

The aim of this work is to develop a context management approach that uses semantics and is tunable even at the run time. The approach will need to be scalable from personal space to home and from office spaces to city environment. Its execution environment is highly dynamic as the information changes constantly, e.g., with new sensors or devices appearing, new temperature sensor output values, and new users or visitors.

As already pointed out in the concept description above, with its separated agents using the semantic database, such as the SIB, the solution is a scalable and reusable one. The agents also need to be scalable, reusable, and configurable. An agent can be, e.g., configured to monitor a certain set of the context. This is called contextual scoping. The scoping is a way to focus on a specific aspect or functionality. CAMA fulfills the gap concerning the self-adaptation of context data processing, which was identified in the recent survey by Bellavista et al. in [20]. Compared to the dedicated solutions that were introduced in Section 1.2, CAMA has the advantage of being independent of context data delivery techniques and implementations languages.

CAMA was compared, in Publication VI, to the middleware classification from the viewpoint of the developer of context-aware applications. The classification was performed by Eugster et al. for 22 middleware platforms [79]: AURA, CAMUS, CARISMA, CARMEN, CASS, CoBrA, ContextToolkit, Cooltown, Context-aware Pub/Sub, CORTEX, CoWSAMI, EgoSpaces, Hydrogen, INFOWARE, LimeLite, MiddleWhere, MobileGaia, MobiPADS, PERVAHO, SOCAM, SpatialViews, and STEAM.

Only few middleware platforms (3 platforms out of 22) were interesting for us due to them both providing high-level programming support and complying with the three given architectural dimensions. The high-level programming support means that the middleware platform adds a context storage and management layer to the previous layers. The three architectural dimensions are: (1) decentralization; (2) portability; and (3) interoperability. Decentralization measures a platform's dependence on specific components. Portability classifies platforms into two groups: portable platforms, which can run on many different operating systems, and operating system dependent platforms, which can only run on few operating systems (usually one). Interoperability measures the ease with which a platform can communicate with heterogeneous software components.

Ideal interoperable platforms can communicate with many different applications, regardless of the operating system they are built on or of the programming language they are written in. The three relevant middleware platforms were CARISMA, Hydrogen, and CoWSAMI. Normally they do not have the ontology support for providing an information-based interoperability as is available in the context management approach of this thesis.

It is a common architectural pattern to simplify the complexity by dividing it according to the characteristics or functionality of the system. According to the author's experience, the division was done in a top-down fashion in the digital BTS,

which is an example of the usual but huge embedded system. In ubiquitous systems the division needs to be done in a bottom-up way, as these systems are more heterogeneous as regards their i) computing resources from sensors to coffee makers and from cars to mobile phones, and ii) user interfaces.

The author claims that the right way to divide the complexity in ubiquitous systems is to use contextual scoping. The context is a well-known and commonly used term in embedded and real-time systems, where it has not been used in a very user-oriented way, as it has been in pervasive or ubiquitous computing. The other difference between these computing areas is to be found in the management of privacy and security. The use of context is discussed in more detail in Publication I.

A typical example of the use of context-awareness agents is given in Figure 12. The numbered arrows between the agents and SIB show a logical work division as the context is usually monitored before the reasoning and context-based adaptation. Publication VI presents the context monitoring, context reasoning, and context-based adaptation agents more deeply.

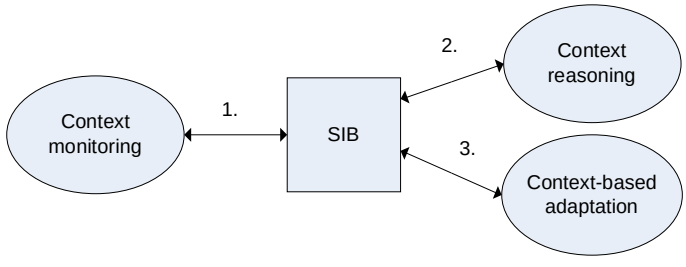

Figure 12. Context-awareness agents and one SIB.

The context monitoring and reasoning agents can be used to communicate with two SIBs, as is exemplified with the context reasoning agent in Figure 13. The context reasoning agent functions as is requested in its rules, and there is a rule also for publishing specific context information to the Home SIB alongside the Personal SIB. This kind of communication is known as cross-domain communication as the dialogue happens with two domains: personal and home space. CAMA thus constitutes the first result of this thesis – the context management approach which uses semantics at the run time.

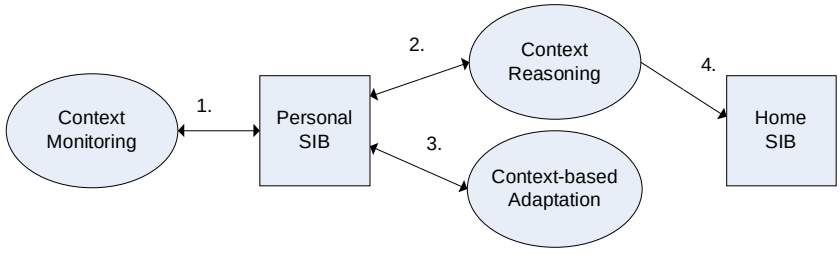

Figure 13. Example of context-awareness agents working with two SIBs.

3.4 Context Ontology for Smart Spaces

This thesis proposes a new context ontology called Context Ontology for Smart Spaces (CO4SS) to be used for situation identification in context-aware applications. This ontology was formulated because existing ontologies do not have the following characteristics:

- support for physical, digital, historical, situational, user, and social context
- the quality management concepts are not mixed to the context ontology
- support for illustrating a situation as a rule or a set of rules.

The core part of SOUPA [71] is very similar to that of CO4SS. SOUPA also includes extension parts for, e.g., supporting specific types of pervasive application domains. Both CO4SS and the 'SOUPA core' use existing ontologies for describing the user and properties needed for pervasive and ubiquitous applications. CO4SS does not include security concepts, unlike the SOUPA core.

CO4SS was enhanced from the context ontology presented earlier within the context-awareness concept. It is more extensive and richer with the concepts and properties used. The user context and the situational context form separate parts and the social context was added after a case study where the context-aware supervision was developed for a smart maintenance process (see Publication IX).

Previously CO4SS employed beliefs, desires and intensions enabled through agents like in the SOUPA core. We do not have them anymore because we decided to use rules for describing desired situations and intended behavior in the smart space. CO4SS uses five contextual levels: physical, digital, situational, user, and social. The historical context can include information from all the levels and, therefore, it cannot be classified as one of the contextual levels.

The main class in CO4SS is the *Context*, including properties for representing *State*, *Time*, and *Location*. The main class is one of the enhancements realized after the first version of context ontology, as is shown in Figure 11. The properties of the Context are passed down to the subclasses, which are: PhysicalContext, DigitalContext, SituationalContext, UserContext, SocialContext, and HistoricalContext. The contextual levels, the main class, and the subclasses are described in more detail in Publication VII.

CO4SS is further enhanced by mapping with a Runtime Performance Management ontology [22] and an Information Security Measuring Ontology (ISMO) [80]. The former mapping is illustrated in Publication III and the latter mapping was originally presented in Publication VIII and, later, as an enhanced version in Publication III. CO4SS is the second result of this thesis – an ontological context model that is generic and suitable for heterogeneous domains.

3.5 Contributions vs. related work

This section introduces a side-by-side comparison of the contributions of this thesis with the related work in the form of a table; two tables have been set up for

CAMA and one for CO4SS. Table 4 presents a comparison of CAMA related to other comparable platforms and solutions. Table 5 introduces how the use of MAPE-K in CAMA differs from the original presentation by IBM and from the use of MAPE-K in autonomic computing and in self-adaptive software. Table 6 presents a comparison of CO4SS related to other context ontologies or ontologies used for adaptations in the application.

CAMA uses a semantic database, such as the SIB, to share the knowledge. This increases modularity and reusability, as the elements in the context management communicate via the SIB and not directly with each other. CAMA is an extension of the IOP [26] and it uses the MAPE-K loop [28], [52] in a new manner. It supports i) interoperability by the semantics of information, and ii) scalability by being modular and decoupled. It performs context management at run time by adapting its behavior according to the situation and supporting other software agents to adapt accordingly. It is tunable at run time and also reusable as such or partly.

Table 4. The contributions of CAMA compared to other platforms or middleware.

Related work	**Differences**
Mitola et al. [10] (cognitive radio)	Similar approach - has a cognition cycle. - has a learning path and software agents. - uses micro-“things”.
Korpipää [14], Boytsov et al. [15] Roy et al. [16], Rothermel et al. [17], Xue et al. [18], Wei et al. [19]	Dedicated solutions - are not as reusable as CAMA - are not independent of o context data delivery techniques o implementation techniques.
Eugster et al. [79] (middleware classification)	Does not have ontology support for providing information-based interoperability.
Bellavista et al. [20] (context data distribution system)	Includes - the context data sources and sinks - the context data security; problematic for modularity and decoupling. Omits scalability. Does not emphasize interoperability. Does not take into account: - Hydra [23], CAMPUS [19] [24], - MUSIC [25], and IOP [26].
Hinchey et al. [51] (DSPL)	Has no support for interoperability. Has minimal or even non-existing continuous support for system changes because of the coming and going of devices, appliances or users Guides toward building applications as a monolith and in a top-down way.

Interoperability platform [26] (IOP)	Has no context-awareness capability.
Zhang et al. [23] (Hydra)	Uses SeMaPS ontologies - supports a dynamic context - no dynamically changing rules. Is dependent on OSGi.
Hallsteinsen et al. [25] (MUSIC)	Uses ontologies only during design-time. Adaptation decisions are based on utility functions. Is dependent on OSGi. Applies the MAPE-K loop.
Bencomo et al. [53] (conceptual model of DSPL)	Applies the MAPE-K loop.

CAMA uses the MAPE-K loop by using a database to share the knowledge. This work brings more granularity to the MAPE-K loop by i) using a semantic database to share the knowledge instead of, e.g., sending messages inside the MAPE-K loop, and ii) decoupling the MAPE parts into separate agents so as to promote scalability. CAMA does not need the Analyze (A) or Plan (P) components of the MAPE-K loop as it operates as a servant for the other parts in the smart application. It requires a reasoning part instead. The author prefers to use the term adaptation instead of execution. Consequently, CAMA was designed to use agents for monitoring, reasoning and adaptation. The semantic database is used as the basis for interoperability. It is used “globally” in the system and not only locally by the proposed approach.

Table 5. Contributions of CAMA compared to MAPE-K and autonomic computing.

Related work	**Differences**
the IBM manifesto [28], the IBM blueprint [52] (MAPE-K loop)	Has control loop with four stages. Uses communication via asynchronous technologies, such as a messaging bus. Has no semantics. Has no database for the K-part. No decoupling as the loop is deployed as one element. Scalability is a challenge because of no decoupling. Reuse is not considered; characteristics are not shared for reusable and system specific ones.
Huebscher et al. [7], (usage of MAPE-K)	The loop is a single software component. The knowledge is shared inside the loop. The loop has three stages. Data flows between the stages until adaptation happens; tight dependency. Mentions semantics as a challenge for the autonomic computing community.

Kephart et al. [30] (vision of autonomic computing) Salehie et al. [31] (self-adaptive sw) Garlan et al. [32] (Rainbow)	The loop has three stages. Data flows between the stages until adaptation happens; tight dependency. Has no database for the data-flow.
Parashar et al. [29] (an overview of autonomic computing)	The loop has two stages. Has no database for the data-flow. Data flows between the stages until adaptation happens; tight dependency.

CO4SS is expandable for heterogeneous domains and run-time quality management. It can be aligned with separate ontologies. It uses concepts, such as software agents, for providing services and rules for defining use situations. It does not include concepts that are application specific or needed for measuring or handling qualities.

CO4SS is composed of smaller ontologies to enable it to be reusable, as such or partly. Thus, it is also generic. It provides the semantics, i.e., the third level of information, cf. Figure 3, and it also enaebles semantic interoperability, cf. Figure 2.

Table 6. Contributions of CO4SS compared to the related work.

Related work	**Differences**
Mitola et al. [10] (cognitive radio)	Has no support for the semantics.
Bellavista et al. [20] (context data distriebution system)	Has no support for the semantics.
Comparison in [69] includes CoOL [70], SOUPA [71], and CONON [72] [73]. It also has CoDAMoS [74].	Not composed of multiple ontologies. Concepts are either application specific or needed for measuring and handling qualities.
Wei et al. [19], [24] (CAMPUS)	The concept of physical quantity is mixed to its context ontology. Has no concepts for situations. Dedicated to work with the CAMPUS middleware. Enables changing a composition at run time.
Zhang et al. [23] (SeMaPS)	Is presented with the Hydra Middleware; OSGi dependency. Is a device oriented set of ontologies. Has no concept of the situation. Includes, e.g., Quality of Service (QoS) ontology. Supports dynamic context. Has no dynamically changing rules.

4. Validating the context-aware micro-architecture

The validation cases carried out will be introduced in this chapter. The first case is concerned with the use of context monitoring to manage information security at the run time. The second case is located in the smart home, where CAMA is used in a cross-domain fashion with CO4SS. In this case the design-time configuration was used. The third case has to do with a smart building maintenance solution, which was enhanced by a context-based adaptation. The third case validates the reconfigurability and reusability of the context monitoring agent. The fourth case introduces the adaptation framework, where CAMA and CO4SS are adapted to be used with the run-time security and performance management to develop situation-based and self-adaptive applications for the smart environment. The fourth case used the run-time configuration.

The validation cases exploited the semantic information according to CO4SS. A dynamic and behavioral interoperability between the receiver(s) and sender(s) of the information was achievable at run time. The context-aware micro-architecture proved to be reusable from case to case entirely or partly. It was used partly – as a context monitoring agent – by the run-time security management, as described in Section 4.1 and Publication VIII. The CAMA agents used the database to communicate with each other or with the rest of the application. Therefore, CAMA is decoupled and it is also modular as its functionality does not include any embedded management of security or any other quality attributes.

4.1 Context monitoring for run-time security management

Context monitoring is used for recognizing the changes in the context information that are relevant for security. It is used as a part of a micro architecture for security adaptation, as presented in Figure 14. The context information for security has originally been presented in Publication VIII. The security is keen to be informed when there are changes in

- the role of exchanged or stored data
- the role of the smart space and its services and repute of the services
- the role of the user in the smart space.

The corresponding concepts and properties of CO4SS used for monitoring the above mentioned context information for security are introduced in Publication III. This case, even at the concept level, showed how easily the context monitoring agent can be reused to produce input for the security management. In a similar way, the context monitoring agent can also be used for fulfilling other input needs.

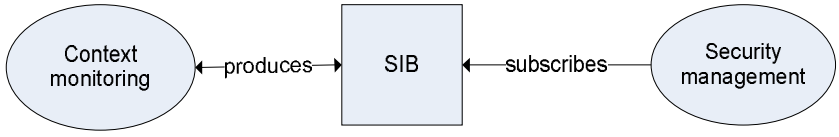

Figure 14. Context monitoring used in micro-architecture for security adaptation.

4.2 Context-aware behavior in the smart home

Two applications were developed to test CAMA and the coverage of CO4SS. These cases behaved according to the design-time configuration, originally presented in Publications VI and VII. The run-time configuration was experimented with within the adaptation framework and is presented in Section 4.4 and in Publication III.

The behavior in the applications was illustrated as Message Sequence Charts (MSCs), which were created with the Microsoft Visio tool. The code for the first application was designed and generated by the Smart Modeller tool, which is intended for end-user programming [81] [82]. The second application, which was concerned with the lighting control system, was implemented with the Python programming language. The latter application used RIBS [57] as a SIB for both home and personal spaces. The intention with this application was to validate the creation of context-awareness agents with the design-time configuration. The applications were decided to be very simple to allow thorough testing of the practical achievements, CAMA and CO4SS.

The first application was designed for

1. Waking up the user, according to the first meeting of the current day (if a working day) and the time that is usually used for the morning activities after wake-up
2. Preparing coffee when the user is woken up.

The latter scenario scaled up from the personal space to a cross-domain application working between the personal and smart home spaces.

The second application was designed to switch on the lighting at the right time, according to the preferences set by the user. The smart home was simulated. The usefulness of the context ontology in carrying out reasoning actions based on the user context was validated in this application. A domain specific ontology was created to control the lights. The desired behavior for the user at home was successfully implemented by using CAMA and CO4SS in both applications.

Both of the applications showed the reusability and the scalability of the semantic context-aware micro-architecture. These applications were invisible, as far as it was possible to work based on the preferences and the calendar of the user.

4.3 Context-based adaptation in smart building maintenance

An existing smart building maintenance application was enhanced with context monitoring to enable it to supervise its progress in a maintenance scenario. The domain specific ontology for building maintenance was already in existence and CO4SS was mapped with it to share the monitored context. This case study was originally presented in Publication IX.

A new way to bond the relevant context information to the context monitoring agent was created, in which the context is selected with a Graphical User Interface (GUI), called Context Selector. The creation of the Context Selector was part of the thesis work of another person and it was presented in Italian in [83]. The Context Selector uses CO4SS and domain specific, i.e., building maintenance related, ontologies in the creation of selection boxes. The user, a maintenance operator in this case, can then select the relevant context by clicking the respective boxes.

The mapping between CO4SS and the building maintenance ontology had to be done by hard coding because the ontologies did not share the same concepts. The context monitor, once again, proved to be a reusable approach. It is also scalable as it can be instantiated multiple times for different monitoring purposes, even within a single application. The behavior of the context monitor is configurable, as in this case via the Context graph.

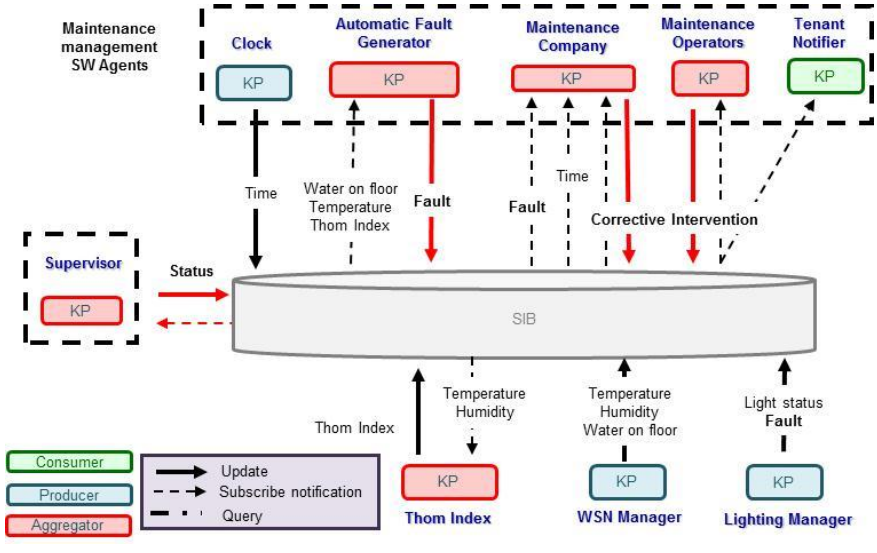

Figure 15. Context-aware Supervision of a Smart Maintenance Process.

Figure 15 illustrates the smart maintenance application that was enhanced by the context-aware supervisor. The supervisor is a context monitoring agent that is instantiated for every fault that appears in the building. The supervisor looks after the building maintenance process to make sure it runs as planned. In case the process is not proceeding the way it should, the maintenance company is informed via the SIB of the delay in the building maintenance.

4.4 Adaptation framework

Context management using dynamic rules is explored in the adaptation framework, where the context management is aligned with the security and performance management. The adaptation framework represents a novel architecture with generic ontologies for context, security, and performance management. It uses dynamic models to perform run-time reasoning and adaptation. By using the framework, the application developer can create situation-based and self-adaptive applications for the smart environment. The framework has been introduced in detail in Publication III.

The adaptation parts of context, security, and performance management are separated from their corresponding micro-architectures and gathered together in the framework to enable managing the adaptation. This forms a centralized element for managing the execution of the adaptation based on the models that are updatable at the run time. Publication III presents, in more detail, the related solutions, the MUSIC [25] and the Hydra [23] middleware, which are compared to the adaptation framework. The adaptation framework differs from MUSIC and Hydra in the management of the adaptation, and in the run-time use of the ontologies. MUSIC and Hydra are both dependent on OSGi, unlike the adaptation framework.

In the adaptation framework the security management, for example, constitutes a separate element with its specific internal parts, which enables it to independently follow the execution of the rest of the MAPE-K loop. The inner parts are coupled together, which is not the case with context building. Context building functions as a core servant for the other run-time management facilities. Therefore its inner parts, which are needed for monitoring, filtering, and reasoning, are working independently and only share the context information via the SIB.

The adaptation framework showed the viability of the dynamic micro-architectures as building blocks when developing situation-based and self-adaptive applications for the smart environment. It also proved to work in an invisible way as it did not disturb the users. Generic ontologies are needed in addition to micro-architectures to support run-time updatable rules and to provide an evolvable semantic interface to different devices for sharing information. This case and the cases described above prove that the third result of this thesis – demonstrations that validate the developed approach and the ontological context model – is accomplished.

5. Discussion and conclusions

5.1 The manifesto of autonomic computing and the information levels

IBM's manifesto [28] has been an inspiring vision to reach a system that computes autonomously. The manifesto provides eight characteristics that are at a high-level of abstraction but relevant for the embedded system, such as the BTS of a mobile network whose duty is to serve the network operator to fulfill the needs of mobile phone users. As stated earlier in the introduction, the work that is described in this thesis concentrates on two of these characteristics: self-awareness and context-awareness. This is due to the experience gathered during the fifteen years at the software development of the BTS in Nokia Siemens Networks. According to this experience the system has to first know itself and its environment to be able to fulfill the other requirements, such as to configure itself, to optimize itself, to recover, and to protect itself. To know its state and its environment the system has be context-aware.

CO4SS provides a common language for designing applications to be able to form an interoperable system that also has a few – if not all – the characteristics of the manifesto. CO4SS enables the semantics, the third level in Gitt's layers of information, which is presented in Figure 3. CO4SS is also modular, as it consists of the ontologies for a physical, digital, social, user, and situational context. It is also expandable by the domain ontologies, as was performed in the smart home applications, described in Section 4.2. To be modular and expandable means that CO4SS provides scalability.

CO4SS can be mapped to the existing ontology, as was done in the smart building maintenance that was highlighted in Section 4.3. It has also been enhanced to work for the run-time information security as presented in Section 4.1. In addition, it has been aligned with the run-time performance management ontology; see Section 4.4 and Publication III. Thus, CO4SS forms the basis for achieving an interoperable system or application.

The ontologies can be seen as interface specifications of the semantic web that is an enhancement of the current World Wide Web (WWW). The WWW is based on the open standards and the same applies to the semantic web. The ultimate

idea with the ontologies in the semantic web is that they are openly usable via the Internet, but they can also be used in a proprietary fashion. The Internet has become the de facto pervasive communications system across the world [84].

The usage of ontologies complies with the seventh, “open”, characteristic of the manifesto. Ontologies are created with a Web Ontology Language (OWL) and the information is saved as RDF triples to the database. In the application cases of this work, the information is exchanged through the SSAP protocol, which is the XML-based interaction protocol. For deduction, we used a Query Language for RDF (SPARQL), as it is a generic solution that can be applied with all application programming languages. Publication III presents the comparison of rule languages. The OWL, RDF, XML, and SPARQL are specifications, recommendations, or standards of the W3C.

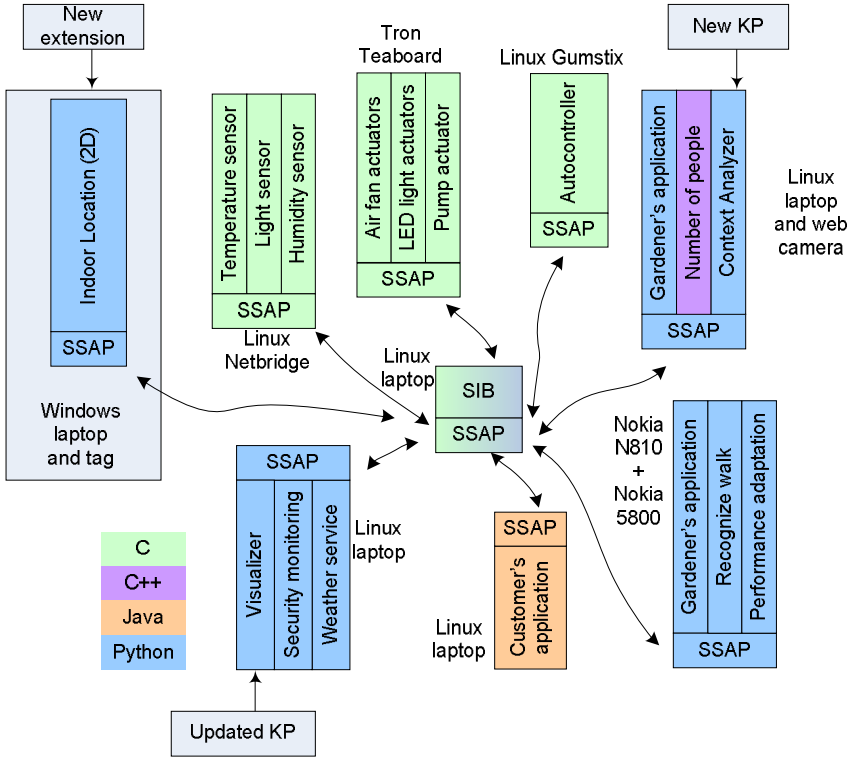

Figure 16. Enhanced Greenhouse Demonstration.

In addition to the open standards and protocol, the CO4SS, like ontologies in general, provide a clear advantage to build a system based on heterogeneous devices or things that are computing applications written with many languages. The legacy devices can also be easily added as part of the system. Publication III goes through the demonstrations done in the SOFIA project by the author and her

colleagues that relate to the adaptation framework which is presented shortly in Section 4.4.

The first demonstration, Smart Greenhouse [84], impressively showed, e.g., how the smart space evolves and extends incrementally without modification to the existing devices. This demonstration was later enhanced by the legacy device to provide an indoor location of the gardener as illustrated in Figure 16. The indoor location information was exploited to detect if or when the Gardener came close to the external display from the Linux laptop to change the computing of the Gardener's application from his portable device, Nokia N810, to the Linux laptop. The legacy device was easily added to the existing demo with a small adapter software that read the location information and wrote it to the SIB.

The second characteristic of the manifesto – self-configuring – is reached by the dynamic agents of CAMA, as they are configurable at run time via the semantic database, such as Smart-M3 SIB or RIBS. The GuideMe application, presented in Publication III, was created by the adaptation framework and the actions of context-reasoning agent were controlled by the dynamically changing rules. A dynamic context monitoring agent was created in the smart building maintenance case as presented shortly in Section 4.2 and in more detail in Publication IX.

CAMA and CO4SS provide the following four characteristics of the manifesto:

1. Self-awareness
2. Self-configuring
3. Context-aware
4. Open.

The remaining characteristics – self-optimizing, self-healing, self-protecting, and anticipatory – are system specific features. Therefore, they cannot be directly provided – even though supported – by the generic facilities offered by CAMA and CO4SS. Scalability is not evidently clear in the manifesto, as it is an invisibility, even though it considered as a "hidden" characteristic. CAMA and CO4SS support both scalability and invisibility. The agents of CAMA are working on the information levels four, pragmatics, and five, apobetics, as presented in Figure 3. They support reaching the targeted aim by monitoring, reasoning and activating required adaptations.

Dobson et al. wrote in 2010 [85] that the original vision of autonomic computing remains unfulfilled. The authors were asking if we engineers had set an impossible goal for ourselves to create a true autonomy from a collection of programs. They also state that while too many solutions to individual problems have been created, there is a need for integrating solutions that combine the point solutions into autonomic systems. CAMA and CO4SS are generic and targeted to provide interoperability between the different actors in a system and, therefore, they possess the required characteristic as an integrating solution for an autonomic and intelligent system.

The paper by Kephart and Chess [30] has been used a great deal as a reference for the MAPE-K loop. The authors approach autonomic computing from the viewpoint of an enterprise system and claim that self-management is the essence

of autonomic computing systems. They concentrate on the four characteristic of the manifesto: self-configuration, self-optimization, self-healing, and self-protecting. They do not concentrate on any other characteristics, as is done in this work. The other difference is that their MAPE-K loop works inside an autonomic element which will be an agent. Their MAPE-K works as a closed loop and inside the agent, whereas in this work the MAPE-K loop is distributed to the different agents which communicate via the semantic database.

5.2 Architecture-based adaptation and adaptation frameworks

Dynamic adaptation models are highlighted by Oreizy et al. in [86]. One of those models is proposed by Kramer and Magee [87] and their model is a layered reference architecture for autonomous or self-managed systems. The reference architecture is inspired by robots and has three layers. The MUSIC development framework [25] presents itself to be compliant with the MAPE-K model [30] and also compares itself to the reference architecture of Kramer and Magee.

Publication III introduces, in more detail, the adaptation framework shortly presented in Section 4.4 and goes through two other context-aware and self-adaptive solutions for networked devices which are closest to the adaptation framework. Those solutions are MUSIC and Hydra [23].

The paper of MUSIC development framework [25] is published at the same time with Publication III. The differences between these frameworks come from the usage of the ontologies and the management of the adaptation. MUSIC only uses ontologies during the design-time and adaptation decisions are based on utility functions. Hydra has the SeMaPS ontologies that support a dynamic context, but not dynamically changing rules. MUSIC and Hydra are both dependent on OSGi, as is not the case with the adaptation framework and its micro-architectures such as CAMA.

5.3 Practical implications and limitations

This thesis introduces a novel approach for a run-time software architecture that is a context-aware micro-architecture, CAMA, which has a core role in managing interoperability and dynamics. CAMA can be seen as a servant for other micro-architectures or functionalities needed. CAMA embodies a new bottom-up way of designing a context-aware application. It is reusable and modular, as it can be used as such or partially by employing, for example, just its monitoring agent. CAMA is a new architectural pattern. It is decoupled as its agents use the database to communicate with each other or with the rest of the application. It supports the semantic approach.

The semantic addition to CAMA is reached by the context ontology for smart spaces, CO4SS. The CO4SS ontology is a new one, it provides a wide coverage and it is composed of physical, digital, situational, user, social, and historical context dimensions. It is a common language for smart space applications, expanda-

ble, and it can be aligned with the domain-specific and quality management ontologies. The work reported in this thesis and the related work in the micro-architectures for the run-time security and performance management has formed the basis for the creation of a reusable monitor agent [40].

Here are some points worth noting with regard to the implementation effects of CAMA and the CO4SS:

- The application developer is free to choose the implementation language and this is also supported by different KPIs that are available at the Smart-M3 Open Source Project, see [55].
- CO4SS is an OWL-file or a composite of the OWL-files.
- The semantic approach requires an RDF-based database, such as the SIBs are.
- The context reasoning is successfully exemplified by using the SPARQL.
- XML-based interaction protocol, SSAP, is used between the agents of CAMA and the SIBs. The SSAP is also available at the Smart-M3 Open Source Project, see [55].
- OWL, RDF, and XML are W3C's recommendations and SPARQL is a W3C's standard.

The power in CAMA relies on the usage of the standard and web-based techniques, in the separation-of-concerns principle, and in the enhanced MAPE-K loop. CAMA is highly dynamic, which is due to the run-time updatable rules. The creation of the rules is slightly laborious and the rules are usually created while describing the functionality of used agents by the MSCs. The rules present the desired behavior and are written into the text boxes of MSCs, as can be seen in Figures 10 and 11 in Publication III. There is a lack of tools for designing this kind of dynamic behavior. Open source tools for editing ontologies exist, such as Protégé, and NeOn toolkit.

CAMA has been validated in two different SIBs, but not with the "Big Data" storage. The scalability seems to be attainable with CAMA through multiplying the amount of agents. The scalability of CAMA is to be validated more thoroughly in the future. CO4SS has the potential to become a de facto context ontology for the context-aware, i.e., intelligent applications.

5.4 Theoretical implications

This thesis provides new knowledge, e.g., by comparing

- the development of the embedded software system to the ubiquitous one in Publication I,
- CAMA and CO4SS to the requirements set for the context distribution system in this thesis,

- the information levels originated from the information theory to the interoperability levels of smart environment,
- CAMA and CO4SS to the characteristics of the IBM's manifesto for the autonomic computing.

CAMA and CO4SS are similar approaches, as presented in the intelligence creation for the cognitive radio in [10] because of the usage of i) the cognition cycle in the cognitive radio and the MAPE-K loop in this thesis, ii) the software agents driven by knowledge, and iii) the micro-worlds in the cognitive radio and the micro-architectures proposed to be used to design applications or systems in this work.

This thesis also highlights how the characteristics of the IBM manifesto for autonomic computing is congruent with the high-level requirements of the BTS as a huge embedded system. The control-loop technique for autonomic computing, MAPE-K, is used in a new way for decoupling the agents by using the semantic database to exchange information between the agents. The usefulness of the DSPL in developing situation-based and self-adaptive applications is also discussed.

5.5 Employment in the other areas

The mobile network is global and, in that sense, it is "ubiquitous". As it is spread around the world, it has the potential to be an enabler for digital services. The author believes that the base stations are viable candidates to be service providers for the smart environment. The base station can offer storage to be used for the semantic database(s). The semantic databases can be used together with base band pools to meet the constantly changing capacity and coverage demands on the mobile network.

The ontologies can be used "privately" within the system, such as the digital BTS, even though they are meant to be openly used via the Internet for all of those that are interested in taking advantage of the ontologies. CAMA and ontologies can be used in the digital BTS:

1. to configure it
2. to develop cognitive radio
3. to activate certain features or services at the run time
4. to manage faults
5. to manage the usage of the memory in the real-time and to control the software. The DSP software has hard real-time requirements. CAMA with the ontologies might meet the requirements in the usage scenarios where the requirements are "only" real-time and not hard real-time.

A smart environment can use the information that is provided by the legacy devices or appliances with the help of software adapters that will do the translation from legacy to the semantic information; cf. Indoor Location system in Figure 16. The

ubiquitous research has neglected the existing mobile network as a building block to reach the Weiser's vision.

5.6 Recommendations for future work

The context-aware ecosystem is missing, e.g., its system architecture can be relieved by using the existing network infrastructure to carry the computing and controlling burden. The 3G and 4G base stations and the resources in the cloud can form the global computing pool for the smart environment. Therefore, it would be interesting to explore the usage of the 3G and 4G base stations as computing platforms for the semantic databases. This may enable new revenue, e.g., for network infrastructure providers and network operators.

As written above, the CAMA, in terms of its scalability, requires further validation. To do that, the following viewpoints are to be taken into account:

- to use the "Big Data" database, as it includes a lot more data than the SIBs that are used during the work of this thesis.
- to multiply the amount of CAMA agents.
- to stress the context reasoning agent with more rules and with more nested rules than is done in the work for this thesis.

6. Summary

Since 1991, when Weiser's vision of the ubiquitous computing arose, there has been a multitude of research results to face certain challenges or to provide other approaches to support pervasiveness. The ubiquitous technologies are reaching everyday life, but they have not yet been brought through. The ecosystems for the ubiquitous appliances are still missing and this is hindering the breakthrough. Ubiquitous ecosystems are hard to establish, as they should cover all the things that we are using, ranging from home appliances to cars, bicycles, and portable devices.

The main contribution of this thesis is the development of a reusable and semantic approach to developing situation-based and self-adaptive software so as to enable our surroundings to behave in an intelligent way. This approach includes CAMA and CO4SS and it is independent of implementation languages and communication techniques. Thus, it provides free hands for application developers to use the most suitable languages and techniques. The CAMA, as an architectural pattern, is usable without its semantic support, the CO4SS. The context monitoring agent of CAMA has been a motivator for the creation of a reusable knowledge processor by the colleagues of the author. The context monitoring agent is used, e.g., for the run-time security management and is applied to supervise the progress of the building maintenance. It is also applied to the adaptation framework.

The validation cases done showed that CAMA is a reusable solution and together with CO4SS it provides the semantics needed for interoperability. CO4SS proved to provide relevant concepts to build the situation-based and self-adaptive software. It can be expanded with the domain-specific and quality ontologies. Even though the approach that is presented in this thesis is validated to be feasible, invisible, and scalable, additional research is needed to validate the scalability of the approach. Based on the scalability research to come, the deployment guidelines can be formulated.

Ubiquitous computing is still missing system architecture that enables forming a robust ecosystem that is also profit-making. Without a viable system architecture, there is a lack of control and this is hindering the breakthrough of ubiquitous computing or intelligent systems. CO4SS has the potential to be either the de facto ontology for creating situation-based and self-adaptive software, or to form the ontology basis for ubiquitous systems. CAMA embodies a highly dynamic ap-

proach to managing the context needed for specific applications or for specific system functionalities, such as the control of energy consumption, information security management, fault management, or performance management.

In addition to the practical results, i.e., CAMA and the CO4SS, this thesis has produced new knowledge by comparing the information levels originated from the information theory with the interoperability levels of smart environment. In addition, this thesis highlights how the characteristics of the IBM manifesto for autonomic computing are congruent with the high-level requirements of the BTS as a huge embedded system. The standardization of the mobile networks has created a basis for building up ecosystems, even though the regulators based on the standardization are still hindering the breakthrough of the technique, such as a cognitive radio for reusing the radio spectrum available. CAMA and CO4SS might be usable in the creation of intelligence for the cognitive radio.

New knowledge is also provided by comparing the proposed approach with the characteristics of the IBM manifesto for autonomic computing as well by comparing the proposed approach with the requirements set for the context data distribution system. The control-loop technique for autonomic computing, MAPE-K, is used in a new way to decouple the agents by using the semantic database to exchange information between the agents. The usefulness of the DSPL in developing situation-based and self-adaptive applications is also discussed.

References

[1] Weiser, M. The Computer for the 21st Century. *Scientific American*, **265**(3) (1991), 94–104.

[2] Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, **8**(4) (2001), 10–17.

[3] Saha, D., Mukherjee, A. Pervasive Computing: A Paradigm for the 21st Century. *Computer*, **36**(3) (2003), 25–31.

[4] Morris, C.W. *Foundations of the Theory of Signs*. International Encyclopedia of Unified Science. Vol.1, No. 2, The University of Chicago Press, originally published in July 1938, 9th impression (1957).

[5] Shannon, C.E., Weaver, W. *The Mathematical Theory of Communication*. University of Illinois Press. Originally published in a clothbound edition in 1949, 6th printing of the paperbound edition (1975). ISBN 0-252-72548-4.

[6] Gitt, W. Information: The Third Fundamental Quantity. *Siemens Review* **6**(89) (1989), 36–41.

[7] Huebscher, M.C., McCann, J.A. A survey of Autonomic Computing – Degrees, Models, and Applications. *ACM Computing Surveys,* **40**(3) (2008), Article 7.

[8] Valenta, V., Marsalek, R., Baudoin, G., Villegas, M., Suarez, M., Robert, F. Survey on Spectrum Utilization in Europe: Measurements, Analyses and Observations. *Proceedings of CROWNCOM 2010. 5th International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, IEEE Computer Society (2010), 1–5.

[9] Haykin, S. Cognitive Radio: Brain-Empowered Wireless Communications. *IEEE Journal on Selected Areas in Communications*, **23**(2) (2005), 201–220.

[10] Mitola, J. III, Maquire, G.Q. Jr.. Cognitive Radio: Making Software Radios More Personal. *IEEE Personal Communications*, **6**(4) (1999), 13–18.

[11] Strang, T., Linnhoff-Popien, C. A Context Modelling Survey, *Proceedings of UbiComp 2004, the 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, (2004), 31–41.

[12] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing*, **6**(2) (2010), 161–180.

[13] Hong, J., Suh, E., Kim, S. Context-aware Systems: A Literature Review and Classification. *Expert System with Applications*, **36**(4) (2009), 8509–8502.

[14] Korpipää, P. *Blackboard-based Software Framework and Tool for Mobile Device Context Awareness*. Ph.D. dissertation, VTT Publications 579, VTT Technical Research Centre of Finland (2005) 225 p. http://www.vtt.fi/inf/pdf/publications/2005/P579.pdf.

[15] Boytsov, A., Zaslavsky, A. Formal Verifications of Context and Situation Models in Pervasive Computing. *Pervasive and Mobile Computing*, **9**(1) (2013), 98–117.

[16] Roy, N., Gu, T., Das, S.K. Supporting Pervasive Computing Applications with Active Context Fusion and Semantic Context Delivery. *Pervasive and Mobile Computing*, **6**(1) (2010), 21–42.

[17] Rothermel, K., Schnitzer, S., Lange, R., Dürr, F., Farrell, T. Context-aware and Quality-aware Algorithms for Efficient Mobile Object Management. *Pervasive and Mobile Computing*, **8**(2) (2012), 131–146.

[18] Xue, W., Pung, H.K., Sen, S. Managing context data for diverse operating spaces. *Pervasive and Mobile Computing*, **9**(1) (2013), 57–75.

[19] Wei, E.J.Y., Chan, A.T.S. CAMPUS: A Middleware for Automated Context-aware Adaptation Decision Making at Run Time. *Pervasive and Mobile Computing*, **9**(1) (2013), 35–56.

[20] Bellavista, P., Corradi, A., Fanelli, M., Foschini, L. A Survey of Context Data Distribution for Mobile Ubiquitous Systems, *ACM Computing Surveys*, **44**(4) (2012), Article No. 24.

[21] Evesti, A., Suomalainen, J., Ovaska, E. Architecture and Knowledge-Driven Self-Adaptive Security in Smart Space. *Computers*, **2**(1) (2013), 34–66.

[22] Purhonen, A., Stenudd, S. Runtime Performance Management of Information Broker-Based Adaptive Applications. In: *Software Architecture*, ECSA 2011, Vol. 6903, I. Crnkovic, V. Gruhn and M. Book, (eds.). Springer-Verlag Berlin and Heidelberg (2011), 203–206.

[23] Zhang, W., Hansen, K.M., Kunz, T. Enhancing Intelligence and Dependability of a Product Line Enabled Pervasive Middleware. *Pervasive and Mobile Computing*, **6**(2) (2010), 198–217.

[24] Wei, E.J.Y., Chan, A.T.S. Semantic Approach to Middleware-driven Runtime Context-aware Adaptation Decision. *Proceedings on the IEEE International Conference on Semantic Computing*, (2008), 440–447.

[25] Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., Papadopoulos, G.A. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *The Journal of Systems and Software*, **85**(12) (2012), 2840–2859.

[26] SOFIA project, http://www.sofia-project.eu, Accessed 03.05.2012.

[27] Liquid Radio – Let traffic waves flow most efficiently. White paper. Accessed 26.4.2012, Available from http://www.nokiasiemensnetworks.com/portfolio/liquidnet.

[28] Horn, P. Autonomic Computing: IBM's perspective on the State of Information Technology. (2001). Accessed 26.4.2012, Available from http://www.research.ibm.com/autonomic/index.html.

[29] Parashar, M., Hariri, S. Autonomic Computing: An Overview. *The Proceedings of UPP2004*, Springer-Verlag, LNCS 3566/2005, (2005), 247–259.

[30] Kephart, J.O., Chess, D.M. The Vision of Autonomic Computing. *Computer*, **36**(1) (2003), 41–50.

[31] Salehie, M., Tahvildari, L. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, **4**(2) (2009), Article 14,1–42.

[32] Garlan, D., Cheng, S-W., Huang, A-C., Schmerl, B., Steenkiste, P. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, **37**(10) (2004), 46–54.

[33] Psaier, H., Dustdar, S. A survey on self-healing systems: approaches and systems. *Computing*, **91**(1) (2011), 43–73.

[34] Parnas, D.L. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, **15**(12) (1972), 1053–1058.

[35] Tarr, P., Ossher, H., Harrison, W., Sutton, S.M.Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. *The Proceedings of ICSE'99*, ACM Press, (1999), 107–119.

[36] Alur, D., Malks, D., Crupi, J. *Core J2EE Patterns: Best Practices and Design Strategies* (2nd Edition) (2003).

[37] Yin, R.K. *Case study research: Design and methods*, 3rd edn. Sage Publications. Thousand Oaks, California, USA (2003). ISBN 978-0-7619-2553-8.

[38] Runeson, P., Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14 (2009) 131–164.

[39] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. *Experimentation in Software Engineering*. Springer-Verlag. Berlin Heidelberg (2012) 55–72. ISBN 978-3-642-29043-5 (Print), 978-3-642-29044-2 (Online).

[40] Kuusijärvi, J., Stenudd, S. Developing Reusable Knowledge Processors for Smart Environments, *Proceedings of SISS 2011, the 2nd International Workshop on Semantic Interoperability for Smart Spaces on the 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011),* (2011), 286–291.

[41] Krueger, C. Software Reuse. *ACM Computing Surveys*, **24**(2) (1992), 131–183.

[42] Frakes, W., Terry, C. Software Reuse: Metrics and Models. *ACM Computing Surveys*, **28**(2) (1996), 415–435.

[43] Pohl, K., Böckle, G., van der Linden, F. *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer-Verlag, Berlin, Germany, (2005).

[44] Bosch, J. *Design and Use of Software Architectures: Adopting and evolving a product-line approach*, Addison-Wesley, Reading, Massachusetts, (2000).

[45] Verlage, M., Kiesgen, T. Five Years of Product Line Engineering in a Small Company, *The Proceedings of the 27th Int'l Conference on Software Eng. (*ICSE'05*)*, ACM Press, (2005), 534–543.

[46] Birk, A., Heller, G., John, I., Schmid, K., von der Massen, T., Muller, K. Product Line Engineering, the State of the Practice. *IEEE Software,* **20**(6) (2003), 52–60.

[47] Clements, P., Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, Reading, Massachusetts, (2002).

[48] Bosch, J. Product-line Architectures in Industry: A Case Study, *The Proceedings of the 21st Int'l Conference on Software Eng. (*ICSE'99*)*, IEEE Computer Society Press, Los Angeles, CA, (1999), 544–554.

[49] Clements, P., Lawrence, J., Northrop, L., McGregor, J. Project Management in a Software Product Line Organization. *IEEE Software*, **22**(5) (2005), 54–62.

[50] Clements, P., Bachman, F., Bass, L., Garland, D., Ivers, J., Little, R. Nord, R., Stafford, J. *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, Boston, Massachusetts, (2003).

[51] Hinchey, M., Park, S., Scmid, K. Building Dynamic Software Product Lines. *Computer*, **45**(10) (2012), 22–26.

[52] An Architectural Blueprint for Autonomic Computing. (2003). Accessed 26.4.2012, Available from http://cs.nju.edu.cn/yangxc/autonomic-computing/ACwpFinal.pdf.

[53] Bencomo, N., Hallsteinsen, S., Santada de Almeida, E. A View of the Dynamic Software Product Line Landscape. *Computer*, **45**(10) (2012), 36–41.

[54] Manola, F., Miller, E., McBride, B. (ed.) RDF primer, W3C Recommendation 10 February 2004, (2004), http://www.w3.org/TR/rdf-primer/.

[55] Smart-M3 Open Source Project, http://sourceforge.net/projects/smart-m3/.

[56] Honkola, J., Laine, H., Brown, R., Tyrkkö, O. Smart-M3 Interoperability Platform, *Proceedings of SISS 2010, 1st International Workshop on Semantic Interoperability for Smart Spaces on IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society (2010), 1041–1046.

[57] Suomalainen, J., Hyttinen, P.,Tarvainen, P. Secure Information Sharing Between Heterogeneous Embedded Devices, *Proceedings of the 4th European Conference on Software Architecture, ECSA 2010 workshops: 1st Int. Workshop on Measurability of Security in Software Architectures (MeSSA 2010)*, ACM, (2010), 205–212.

[58] Peristeras, V., Tarabanis, K. The connection, communication, consolidation, collaboration interoperability framework (C4IF) for information systems interoperability, IBIS – Interoperability in Business Information Systems **1**(1) (March 2006), 61–72.

[59] Dey, A.K., Abowd, G.D. *Towards a Better Understanding of Context and Context-Awareness*. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, USA, (1999).

[60] Truong, H., Dustdar, S. A Survey on Context-aware Web Service Systems. *International Journal of Web Information Systems*, **5**(1) (2009), 5–31.

[61] Soylu, A., De Causmaecker, P., Desmet, P. Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. *Journal of Software*, **4**(9) (2009), 992–1013.

[62] Kapitsaki, G.M., Prezerakos, G.N., Tselikas, N.D., Venieris, I.S. Context-aware Service Engineering: A Survey. *The Journal of Systems and Software*, 82 (2009),1285–1297.

[63] Achillelos, A., Yang, K., Georgalas, N. Context Modelling and a Context-aware Framework for Pervasive Service Creation: A model-driven approach. Pervasive and Mobile Computing, **6**(2) (2010), 281–296.

[64] Ye, J., Dobson, S., McKeever, S. Situation Identification Techniques in Pervasive Computing: a Review. *Pervasive and Mobile Computing*, **8**(1) (2012), 36–66.

[65] Dhaussy, P., Roger, J-C., Boniol, F. Context Aware Model-Checking for Embedded Software. In: *Embedded System – Theory and Design Methodology.* Tanaka, K. (eds.). InTech. Rijeka, Croatia (2012), 167−184. ISBN 979-953-307-580-7.

[66] Baldauf, M., Dustdar, S., Rosenberg, F. A Survey on Context-aware Systems, *International Journal of Ad Hoc and Ubiquitous Computing*, **2**(4) (2007), 263–277.

[67] Gruber, T. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, **5**(2) (1993), 199–220.

[68] Zhou,J. Knowledge Dichotomy and Semantic Knowledge Management, *Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Application of Semantic Web,* Springer-Verlag, Vol.188 (2005), 305–316.

[69] Ye, J., Coyle, L., Dobson, S., Nixon, P. Ontology-based Models in Pervasive Computing Systems. *The Knowledge Engineering Review*, **22**(4) (2007), 315–347.

[70] Strang, T., Linnhoff-Popien, C., Frank, K. CoOL: A Context Ontology Language to enable Contextual Interoperability. In Stefani, J.-B., Dameure, I. and Hagimont, D.(eds.). *Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, LNCS 2893, Springer Verlag, (2003), 236–247.

[71] Chen, H., Finin, T., Joshi, A. The SOUPA Ontology for Pervasive Computing. In: *Ontologies for Agents: Theory and Experiences*. Tamma, V., Cranefield, S., Finin, T.W., Willmott, S. (eds.). Whitestein Series in Software Agent Technologies and Autonomic Computing, Birkhäuser Verlag. Basel, Switzerland (2005), 233–258. ISBN 3-7643-7237-0.

[72] Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q. An Ontology-based Context Model in Intelligent Environments. *Proceedings of the Communication*

Networks and Distributed Systems Modeling and Simulation Conference (CNDS'04), Society for Computer Simulation, (2004), 270–275.

[73] Wang, X.H., Gu, T., Zhang, D.Q., Pung, H.K. Ontology Based Context Modeling and Reasoning Using OWL. *Proceedings of the Workshop on Context Modeling and Reasoning (CoMoRea'04)*, (2004), 18–22.

[74] Preuveneers, D. Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., De Bosschere, K. Towards an Extensible Context Ontology for Ambient Intelligence. *Proceedings of the 2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, (2004), 148–159.

[75] Brickley,D., Miller, L. FOAF Vocalubary Specification 0.98. Namespace Document 9 August 2010. (2010), http://xmlns.com/foaf/spec/20100809.html.

[76] OSGi Alliance. OSGi Platform – Service Compendium, Technical Report Release 4, Version 4.1, OSGi (2007).

[77] Khushraj, D., Lassila, O., Finin, T. sTuples: Semantic tuple spaces. In: *Proc. of the 1st Annual Int. Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004)*, IEEE Computer Society (2004), 268−277.

[78] Bouillet, E., Feblowitz, M., Zhen Liu, Ranganathan, A., Riabov, A. Semantic Models for Ad Hoc Interactions in Mobile, Ubiquitous Environments. In: *Proc. of IEEE Int. Conference on Semantic Computing (ICSC 2008)*, IEEE Computer Society (2008), 589–596.

[79] Eugster, P.T., Garbinato, B., Holzer, A.. Middleware Support for Context-aware Applications. In: *Middleware for Network Eccentric and Mobile Applications.* Garbinato, B., Miranda, H., Rodrigues, L. (eds.). Springer-Verlag. Berlin Heidelberg (2009), 305–322. ISBN 978-3-540-89706-4.

[80] Evesti, A., Savola, R., Ovaska, E. The Design, Instantiation, and Usage of Information Security Measuring Ontology. In: *Proc. of the 2nd Int. Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS 2011)*, IARIA (2011), 1−9.

[81] Katasonov, A. Enabling non-programmers to develop smart environment applications. In: *Proc. of the 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010) in IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society (2010), 1059−1064.

[82] Katasonov, A., Palviainen, M. Towards ontology-driven development of applications for smart environments. In: *Proc. of the 8th PERCOM Workshops*, IEEE Press (2010), 696−701.

[83] Ferrari, S. *Gestione del contesto su architettura Smart M3*, Tesi di Laurea in Calcolatori Elettronici L-A (Sessione III, Anno Accademico 2009/10), University of Bologna, Italy, (2011).

[84] Evesti, A., Eteläperä, M., Kiljander, J., Kuusijärvi, J., Purhonen, A., Stenudd, S. Semantic Information Interoperability in Smart Spaces. in: *Proc. of the 8th Int. Conference on Mobile and Ubiquitous Multimedia (MUM'09)*, ACM International Conference Proceeding Series (2009), 158−159.

[85] Dobson, S., Sterritt, R., Nixon, P., Hinchey, M. Fulfilling the vision of autonomic computing. *Computer*, **43**(1) (2010), 35–41.

[86] Oreizy, P., Medvidovic, N., Taylor, R.N. Runtime Software Adaptation: Framework, Approaches, and Styles. In: *Proc. of the Int. Conference on Software Engineering (ICSE´08)*, ACM International Conference Proceeding Series (2008), 899−909.

[87] Kramer, J., Magee, J. Self-Managed Systems: an Architectural Challenge. In: *Proc. of the Conference on Future of Software Engineering (FOSE'07)*, IEEE Computer Society Press (2007), 259−268.

Publications IV, VI, VII, VIII are not included in the PDF version. Please order the printed version to get the complete publication (http://www.vtt.fi/publications/index.jsp).

PUBLICATION I

Architecting embedded software for context-aware systems

In: Embedded Systems – Theory and Design Methodology. Pp. 123–142.

6

Architecting Embedded Software for Context-Aware Systems

Susanna Pantsar-Syväniemi
VTT Technical Research Centre of Finland
Finland

1. Introduction

During the last three decades the architecting of embedded software has changed by i) the ever-enhancing processing performance of processors and their parallel usage, ii) design methods and languages, and iii) tools. The role of software has also changed as it has become a more dominant part of the embedded system. The progress of hardware development regarding size, cost and energy consumption is currently speeding up the appearance of smart environments. This necessitates the information to be distributed to our daily environment along with smart, but separate, items like sensors. The cooperation of the smart items, by themselves and with human beings, demands new kinds of embedded software.

The architecting of embedded software is facing new challenges as it moves toward smart environments where physical and digital environments will be integrated and interoperable. The need for human beings to interact is decreasing dramatically because digital and physical environments are able to decide and plan behavior by themselves in areas where functionality currently requires intervention from human beings, such as showing a barcode to a reader in the grocery store. The smart environment, in our mind, is not exactly an Internet of Things (IoT) environment, but it can be. The difference is that the smart environment that we are thinking of does not assume that all tiny equipment is able to communicate via the Internet. Thus, the smart environment is an antecedent for the IoT environment.

At the start of the 1990s, hardware and software co-design in real time and embedded systems were seen as complicated matters because of integration of different modeling techniques in the co-design process (Kronlöf, 1993). In the smart environment, the co-design is radically changing, at least from the software perspective. This is due to the software needing to be more and more intelligent by, e.g., predicting future situations to offer relevant services for human beings. The software needs to be interoperable, as well as scattered around the environment, with devices that were previously isolated because of different communication mechanisms or standards.

Research into pervasive and ubiquitous computing has been ongoing for over a decade, providing many context-aware systems and a multitude of related surveys. One of those surveys is a literature review of 237 journal articles that were published between 2000 and

2007 (Hong et al., 2009). The review presents that context-aware systems i) are still developing in order to improve, and ii) are not fully implemented in real life. It also emphasizes that context-awareness is a key factor for new applications in the area of ubiquitous computing, i.e., pervasive computing. The context-aware system is based on pervasive or ubiquitous computing. To manage the complexity of pervasive computing, the context-aware system needs to be designed in new way—from the bottom up—while understanding the eligible ecosystem, and from small functionalities to bigger ones. The small functionalities are formed up to the small architectures, micro-architectures. Another key issue is to reuse the existing, e.g., communication technologies and devices, as much as possible, at least at the start of development, to minimize the amount of new things.

To get new perspective on the architecting of context-aware systems, Section two introduces the major factors that have influenced the architecting of embedded and real-time software for digital base stations, as needed in the ecosystem of the mobile network. This introduction also highlights the evolution of the digital base station in the revolution of the Internet. The major factors are standards and design and modeling approaches, and their usefulness is compared for architecting embedded software for context-aware systems. The context of pervasive computing calms down when compared to the context of digital signal processing software as a part of baseband computing which is a part of the digital base station. It seems that the current challenges have similarities in both pervasive and baseband computing. Section two is based on the experiences gathered during software development at Nokia Networks from 1993 to 2008 and subsequently in research at the VTT Technical Research Centre of Finland. This software development included many kinds of things, e.g., managing the feature development of subsystems, specifying the requirements for the system and subsystem levels, and architecting software subsystems. The research is related to enable context-awareness with the help of ontologies and unique micro-architecture.

Section three goes through the main research results related to designing context-aware applications for smart environments. The results relate to context modeling, storing, and processing. The latter includes a new solution, a context-aware micro-architecture (CAMA), for managing context when architecting embedded software for context-aware systems. Section four concludes this chapter.

2. Architecting real-time and embedded software in the 1990s and 2000s

2.1 The industrial evolution of the digital base station

Figure 1 shows the evolution of the Internet compared with a digital base station (the base station used from now on) for mobile networks. It also shows the change from proprietary interfaces toward open and Internet-based interfaces. In the 1990s, the base station was not built for communicating via the Internet. The base station was isolated in the sense that it was bound to a base station controller that controlled a group of base stations. That meant that a customer was forced to buy both the base stations and the base station controller from the same manufacturer.

In the 2000s, the industrial evolution brought the Internet to the base station and it opened the base station for module business by defining interfaces between modules. It also

dissolved the "engagement" between the base stations and their controllers as it moved from the second generation mobile network (2G) to third one (3G). Later, the baseband module of the base station was also reachable via the Internet. In the 2010s, the baseband module will go to the cloud to be able to meet the constantly changing capacity and coverage demands on the mobile network. The baseband modules will form a centralized baseband pool. These demands arise as smartphone, tablet and other smart device users switch applications and devices at different times and places (Nokia Siemens Networks, 2011).

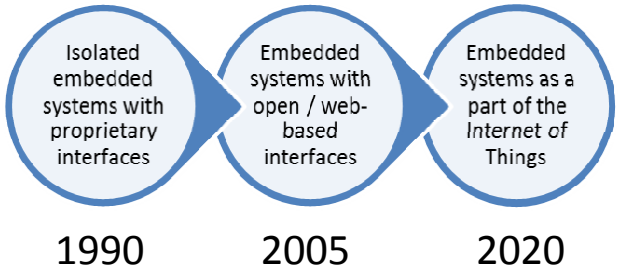

Fig. 1. The evolution of the base station.

The evolution of base-band computing in the base station changes from distributed to centralized as a result of dynamicity. The estimation of needed capacity per mobile user was easier when mobiles were used mainly for phone calls and text messaging. The more fancy features that mobiles offer and users demand, the harder it is to estimate the needed base-band capacity.

The evolution of the base station goes hand-in-hand with mobile phones and other network elements, and that is the strength of the system architecture. The mobile network ecosystem has benefited a lot from the system architecture of, for example, the Global System for Mobile Communications (GSM). The context-aware system is lacking system architecture and that is hindering its breakthrough.

2.2 The standardization of mobile communication

During the 1980s, European telecommunication organizations and companies reached a common understanding on the development of a Pan-European mobile communication standard, the Global System for Mobile Communications (GSM), by establishing a dedicated organization, the European Telecommunications Standards Institute (ETSI, www.etsi.org), for the further evolvement of the GSM air-interface standard. This organization has produced the GSM900 and 1800 standard specifications (Hillebrand, 1999). The development of the GSM standard included more and more challenging features of standard mobile technology as defined by ETSI, such as High Speed Circuit Switched Data (HSCSD), General Packet Radio Service (GPRS), Adaptive Multirate Codec (AMR), and Enhanced Data rates for GSM Evolution (EDGE) (Hillebrand, 1999).

The Universal Mobile Telecommunication System (UMTS) should be interpreted as a continuation of the regulatory regime and technological path set in motion through GSM, rather than a radical break from this regime. In effect, GSM standardization defined a path of progress through GPRS and EDGE toward UMTS as the major standard of 3G under the 3GPP standardization organization (Palmberg & Martikainen, 2003). The technological path from GSM to UMTS up to LTE is illustrated in Table 1. High-Speed Downlink Packet Access (HSDPA) and High-Speed Uplink Packet Access (HSUPA) are enhancements of the UMTS to offer a more interactive service for mobile (smartphone) users.

GSM -> HSCD, GPRS, AMR, EDGE	UMTS -> HSDPA, HSUPA	LTE
2G =>	3G =>	4G

Table 1. The technological path of the mobile communication system

It is remarkable that standards have such a major role in the telecommunication industry. They define many facts via specifications, like communication between different parties. The European Telecommunications Standards Institute (ETSI) is a body that serves many players such as network suppliers and network operators. Added to that, the network suppliers have created industry forums: OBSAI (Open Base Station Architecture Initiative) and CPRI (Common Public Radio Interface). The forums were set up to define and agree on open standards for base station internal architecture and key interfaces. This, the opening of the internals, enabled new business opportunities with base station modules. Thus, module vendors were able to develop and sell modules that fulfilled the open, but specified, interface and sell them to base station manufacturers. In the beginning the OBSAI was heavily driven by Nokia Networks and the CPRI respectively by Ericsson. Nokia Siemens Networks joined CPRI when it was merged by Nokia and Siemens.

The IoT ecosystem is lacking a standardization body, such as ETSI has been for the mobile networking ecosystem, to create the needed base for the business. However, there is the Internet of Things initiative (IoT-i), which is working and attempting to build a unified IoT community in Europe, www.iot-i.eu.

2.3 Design methods

The object-oriented approach became popular more than twenty years ago. It changed the way of thinking. Rumbaugh et al. defined object-oriented development as follows, i) it is a conceptual process independent of a programming language until the final stage, and ii) it is fundamentally a new way of thinking and not a programming technique (Rumbaugh et al., 1991). At the same time, the focus was changing from software implementation issues to software design. In those times, many methods for software design were introduced under the Object-Oriented Analysis (OOA) method (Shlaer & Mellor, 1992), the Object-Oriented Software Engineering (OOSE) method (Jacobson et al., 1992), and the Fusion method (Coleman et al., 1993). The Fusion method highlighted the role of entity-relationship graphs in the analysis phase and the behavior-centered view in the design phase.

The Object Modeling Technique (OMT) was introduced for object-oriented software development. It covers the analysis, design, and implementation stages but not integration and maintenance. The OMT views a system via a model that has two dimensions (Rumbaugh et al., 1991). The first dimension is viewing a system: the object, dynamic, or

functional model. The second dimension represents a stage of the development: analysis, design, or implementation. The object model represents the static, structural, "data" aspects of a system. The dynamic model represents the temporal, behavioral, "control" aspects of a system. The functional model illustrates the transformational, "function" aspects of a system. Each of these models evolves during a stage of development, i.e. analysis, design, and implementation.

The OCTOPUS method is based on the OMT and Fusion methods and it aims to provide a systematic approach for developing object-oriented software for embedded real-time systems. OCTOPUS provides solutions for many important problems such as concurrency, synchronization, communication, interrupt handling, ASICs (application-specific integrated circuit), hardware interfaces and end-to-end response time through the system (Awad et al., 1996). It isolates the hardware behind a software layer called the hardware wrapper. The idea for the isolation is to be able to postpone the analysis and design of the hardware wrapper (or parts of it) until the requirements set by the proper software are realized or known (Awad et al., 1996).

The OCTOPUS method has many advantages related to the system division of the subsystems, but without any previous knowledge of the system under development the architect was able to end up with the wrong division in a system between the controlling and the other functionalities. Thus, the method was dedicated to developing single and solid software systems separately. The OCTOPUS, like the OMT, was a laborious method because of the analysis and design phases. These phases were too similar for there to be any value in carrying them out separately. The OCTOPUS is a top-down method and, because of that, is not suitable to guide bottom-up design as is needed in context-aware systems.

Software architecture started to become defined in the late 1980s and in the early 1990s. Mary Shaw defined that i) architecture is design at the level of abstraction that focuses on the patterns of system organization which describe how functionality is partitioned and the parts are interconnected and ii) architecture serves as an important communication, reasoning, analysis, and growth tool for systems (Shaw, 1990). Rumbaugh et al. defined software architecture as the overall structure of a system, including its partitioning into subsystems and their allocation to tasks and processors (Rumbaugh et al., 1991). Figure 2 represents several methods, approaches, and tools with which we have experimented and which have their roots in object-oriented programming.

For describing software architecture, the 4+1 approach was introduced by Philippe Krüchten. The 4+1 approach has four views: logical, process, development and physical. The last view, the +1 view, is for checking that the four views work together. The checking is done using important use cases (Krüchten, 1995). The 4+1 approach was part of the foundation for the Rational Unified Process, RUP. Since the introduction of the 4+1 approach software architecture has had more emphasis in the development of software systems. The most referred definition for the software architecture is the following one:

> The structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them, (Bass et al., 1998)

Views are important when documenting software architecture. Clements et al. give a definition for the view: "A view is a representation of a set of system elements and the

relationships associated with them". Different views illustrate different uses of the software system. As an example, a layered view is relevant for telling about the portability of the software system under development (Clements, 2003). The views are presented using, for example, UML model elements as they are more descriptive than pure text.

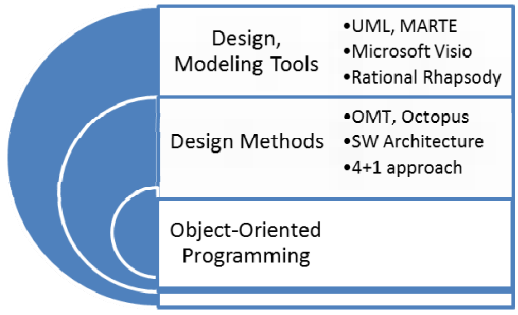

Fig. 2. From object-oriented to design methods and supporting tools.

Software architecture has always has a role in base station development. In the beginning it represented the main separation of the functionalities, e.g. operation and maintenance, digital signal processing, and the user interface. Later on, software architecture was formulated via architectural views and it has been the window to each of these main functionalities, called software subsystems. Hence, software architecture is an efficient media for sharing information about the software and sharing the development work, as well.

2.4 Modeling

In the model-driven development (MDD) vision, models are the primary artifacts of software development and developers rely on computer-based technologies to transform models into running systems (France & Rumpe, 2007). The Model-Driven Architecture (MDA), standardized by the Object Management Group (OMG, www.omg.org), is an approach to using models in software development. MDA is a known technique of MDD. It is meant for specifying a system independently of the platform that supports it, specifying platforms, choosing a particular platform for the system, and transforming the system specification into a particular platform. The three primary goals of MDA are portability, interoperability and reusability through the architectural separation of concerns (Miller & Mukerji, 2003).

MDA advocates modeling systems from three viewpoints: computational-independent, platform-independent, and platform-specific viewpoints. The computational-independent viewpoint focuses on the environment in which the system of interest will operate in and on the required features of the system. This results in a computation-independent model (CIM). The platform-independent viewpoint focuses on the aspects of system features that are not likely to change from one platform to another. A platform-independent model (PIM) is used to present this viewpoint. The platform-specific viewpoint provides a view of a system in which platform-specific details are integrated with the elements in a PIM. This view of a system is described by a platform-specific model (PSM), (France & Rumpe, 2007).

The MDA approach is good for separating hardware-related software development from the application (standard-based software) development. Before the separation, the maintenance of hardware-related software was done invisibly under the guise of application development. By separating both application- and hardware-related software development, the development and maintenance of previously invisible parts, i.e., hardware-related software, becomes visible and measurable, and costs are easier to explicitly separate for the pure application and the hardware-related software.

Two schools exist in MDA for modeling languages: the Extensible General-Purpose Modeling Language and the Domain Specific Modeling Language. The former means Unified Modeling Language (UML) with the possibility to define domain-specific extensions via profiles. The latter is for defining a domain-specific language by using meta-modeling mechanisms and tools. The UML has grown to be a de facto industry standard and it is also managed by the OMG. The UML has been created to visualize object-oriented software but also used to clarify the software architecture of a subsystem that is not object-oriented.

The UML is formed based on the three object-oriented methods: the OOSE, the OMT, and Gary Booch's Booch method. A UML profile describes how UML model elements are extended using stereotypes and tagged values that define additional properties for the elements (France & Rumpe, 2007). A Modeling and Analysis of Real-Time Embedded Systems (MARTE) profile is a domain-specific extension for UML to model and analyze real time and embedded systems. One of the main guiding principles for the MARTE profile (www.omgmarte.org) has been that it should support independent modeling of both software or hardware parts of real-time and embedded systems and the relationship between them. OMG's Systems Modeling Language (SysML, www.omgsysml.org) is a general-purpose graphical modeling language. The SysML includes a graphical construct to represent text-based requirements and relate them to other model elements.

Microsoft Visio is usually used for drawing UML–figures for, for example, software architecture specifications. The UML–figures present, for example, the context of the software subsystem and the deployment of that software subsystem. The MARTE and SysML profiles are supported by the Papyrus tool. Without good tool support the MARTE profile will provide only minimal value for embedded software systems.

Based on our earlier experience and the MARTE experiment, as introduced in (Pantsar-Syväniemi & Ovaska, 2010), we claim that MARTE is not as applicable to embedded systems as base station products. The reason is that base station products are dependent on long-term maintenance and they have a huge amount of software. With the MARTE, it is not possible to i) model a greater amount of software and ii) maintain the design over the years. We can conclude that the MARTE profile has been developed from a hardware design point of view because software reuse seems to have been neglected.

Many tools exist, but we picked up on Rational Rhapsody because we have seen it used for the design and code generation of real-time and embedded software. However, we found that the generated code took up too much of the available memory, due to which Rational Rhapsody was considered not able to meet its performance targets. The hard real-time and embedded software denotes digital signal processing (DSP) software. DSP is a central part of the physical layer baseband solutions of telecommunications (or mobile wireless) systems, such as mobile phones and base stations. In general, the functions of the physical

layer have been implemented in hardware, for example, ASIC (application-specific integrated circuits), and FPGA (field programmable gate arrays), or near to hardware (Paulin et al., 1997), (Goossens et al., 1997).

Due to the fact that Unified Modeling Language (UML) is the most widely accepted modeling language, several model-driven approaches have emerged (Kapitsaki et al., 2009), (Achillelos et al., 2010). Typically, these approaches introduce a meta-model enriched with context-related artifacts, in order to support context-aware service engineering. We have also used UML for designing the collaboration between software agents and context storage during our research related to the designing of smart spaces based on the ontological approach (Pantsar-Syväniemi et al., 2011a, 2012).

2.5 Reuse and software product lines

The use of C language is one of the enabling factors of making reusable DSP software (Purhonen, 2002). Another enabling factor is more advanced tools, making it possible to separate DSP software development from the underlying platform. Standards and underlying hardware are the main constraints for DSP software. It is essential to note that hardware and standards have different lifetimes. Hardware evolves according to 'Moore's Law' (Enders, 2003), according to which progress is much more rapid than the evolution of standards. From 3G base stations onward, DSP software has been reusable because of the possibility to use C language instead of processor-specific assembly language. The reusability only has to do with code reuse, which can be regarded as a stage toward overall reuse in software development, as shown in Figure 3.

Regarding the reuse of design outputs and knowledge, it was the normal method of operation at the beginning of 2G base station software developments and was not too tightly driven by development processes or business programs. We have presented the characteristics of base station DSP software development in our previous work (Pantsar-Syväniemi et al., 2006) that is based on experiences when working at Nokia Networks. That work introduces the establishment of reuse actives in the early 2000s. Those activities were development 'for reuse' and development 'with reuse'. 'For reuse' means development of reusable assets and 'with reuse' means using the assets in product development or maintenance (Karlsson, 1995).

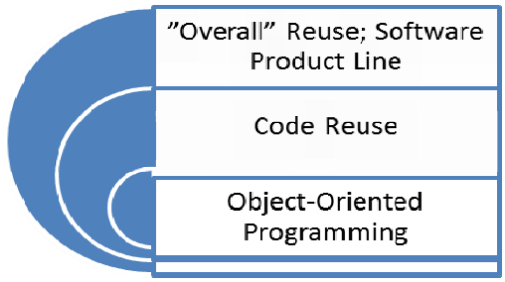

Fig. 3. Toward the overall reuse in the software development.

The main problem within this process-centric, 'for reuse' and 'with reuse', development was that it produced an architecture that was too abstract. The reason was that the domain was too wide, i.e., the domain was base station software in its entirety. In addition to that, the software reuse was "sacrificed" to fulfill the demand to get a certain base station product market-ready. This is paradoxical because software reuse was created to shorten products' time-to-market and to expand the product portfolio. The software reuse was due to business demands.

In addition to Karlsson's 'for and with reuse' book, we highlight two process-centric reuse books among many others. To design and use software architectures is written by Bosch (Bosch, 2000). This book has reality aspects when guiding toward the selection of a suitable organizational model for the software development work that was meant to be built around software architecture. In his paper, (Bosch, 1999), Bosch presents the main influencing factors for selecting the organization model: geographical distribution, maturity of project management, organizational culture, and the type of products. In that paper, he stated that a software product built in accordance with the software architecture is much more likely to fulfill its quality requirements in addition to its functional requirements.

Bosch emphasized the importance of software architecture. His software product line (SPL) approach is introduced according to these phases: development of the architecture and component set, deployment through product development and evolution of the assets (Bosch, 2000). He presented that not all development results are sharable within the SPL but there are also product-specific results, called artifacts.

The third interesting book introduces the software product line as compared to the development of a single software system at a time. This book shortly presents several ways for starting software development according to the software product line. It is written by Pohl et al. (Pohl et al., 2005) and describes a framework for product-line engineering. The book stresses the key differences of software product-line engineering in comparison with single-software system development:

- The need for two distinct development processes: domain engineering and application engineering. The aim of the domain-engineering process is to define and realize the commonality and the variability of the software product line. The aim of the application-engineering process is to derive specific applications by exploiting the variability of the software product line.
- The need to explicitly define and manage variability: During domain engineering, variability is introduced in all domain engineering artifacts (requirements, architecture, components, test cases, etc.). It is exploited during application engineering to derive applications tailored to the specific needs of different customers.

A transition from single-system development to software product-line engineering is not easy. It requires investments that have to be determined carefully to get the desired benefits (Pohl et al., 2005). The transition can be introduced via all of its aspects: process, development methods, technology, and organization. For a successful transition, we have to change all the relevant aspects, not just some of them (Pohl et al., 2005). With the base station products, we have seen that a single-system development has been powerful when products were more hardware- than software-oriented and with less functionality and complexity. The management aspect, besides the development, is taken into account in the

product line but how does it support long-life products needing maintenance over ten years? So far, there is no proposal for the maintenance of long-life products within the software product line. Maintenance is definitely an issue to consider when building up the software product line.

The strength of the software product line is that it clarifies responsibility issues in creating, modifying and maintaining the software needed for the company's products. In software product-line engineering, the emphasis is to find the commonalities and variabilities and that is the huge difference between the software product-line approach and the OCTOPUS method. We believe that the software product-line approach will benefit if enhanced with a model-driven approach because the latter strengthens the work with the commonalities and variabilities.

Based on our experience, we can identify that the software product-line (SPL) and model-driven approach (MDA) alike are used for base station products. Thus, a combination of SPL and MDA is good approach when architecting huge software systems in which hundreds of persons are involved for the architecting, developing and maintaining of the software. A good requirement tool is needed to keep track of the commonalities and variabilities. The more requirements, the more sophisticated tool should be with the possibility to tag on the requirements based on the reuse targets and not based on a single business program.

The SPL approach needs to be revised for context-aware systems. This is needed to guide the architecting via the understanding of an eligible ecosystem toward small functionalities or subsystems. Each of these subsystems is a micro-architecture with a unique role. Run-time security management is one micro-architecture (Evesti & Pantsar-Syväniemi, 2010) that reuses context monitoring from the context-awareness micro-architecture, CAMA (Pantsar-Syväniemi et al., 2011a). The revision needs a new mindset to form reusable micro-architectures for the whole context-aware ecosystem. It is good to note that micro-architectures can differ in the granularity of the reuse.

2.6 Summary of section 2

The object-oriented methods, like Fusion, OMT, and OCTOPUS, were dedicated for single-system development. The OCTOPUS was the first object-oriented method that we used for an embedded system with an interface to the hardware. Both the OCTOPUS and the OMT were burdening the development work with three phases: object-oriented analysis (OOA) object-oriented design (OOD), and implementation. The OOD was similar to the implementation. In those days there was a lack of modeling tools. The message sequence charts (MSC) were done with the help of text editor.

When it comes to base station development, the software has become larger and more complicated with the new features needed for the mobile network along with the UML, the modeling tools supporting UML, and the architectural views. Thus, software development is more and more challenging although the methods and tools have become more helpful. The methods and tools can also hinder when moving inside the software system from one subsystem to another if the subsystems are developed using different methods and tools.

Related to DSP software, the tight timing requirements have been reached with optimized C-code, and not by generating code from design models. Thus, the code generators are too

ineffective for hard real time and embedded software. One of the challenges in DSP software is the memory consumption because of the growing dynamicity in the amount of data that flows through mobile networks. This is due to the evolution of mobile network features like HSDPA and HSUPA that enable more features for mobile users. The increasing dynamicity demands simplification in the architecture of the software system. One of these simplifications is the movement from distributed baseband computing to centralized computing.

Simplification has a key role in context-aware computing. Therefore, we recall that by breaking the overall embedded software architecture into smaller pieces with specialized functionality, the dynamicity and complexity can be dealt with more easily. The smaller pieces will be dedicated micro-architectures, for example, run-time performance or security management. We can see that in smart environments the existing wireless networks are working more or less as they currently work. Thus, we are not assuming that they will converge together or form only one network. By taking care of and concentrating the data that those networks provide or transmit, we can enable the networks to work seamlessly together. Thus, the networks and the data they carry will form the basis for interoperability within smart environments. The data is the context for which it has been provided. Therefore, the data is in a key position in context-aware computing.

The MSC is the most important design output because it visualizes the collaboration between the context storage, context producers and context consumers. The OCTOPUS method is not applicable but SPL is when revised with micro-architectures, as presented earlier. The architecting context-aware systems need a new mindset to be able to i) handle dynamically changing context by filtering to recognize the meaningful context, ii) be designed bottom-up, while keeping in mind the whole system, and iii) reuse the legacy systems with adapters when and where it is relevant and feasible.

3. Architecting real-time and embedded software in the smart environment

Context has always been an issue but had not been used as a term as widely with regard to embedded and real-time systems as it has been used in pervasive and ubiquitous computing. Context was part of the architectural design while we created architectures for the subsystem of the base station software. It was related to the co-operation between the subsystem under creation and the other subsystems. It was visualized with UML figures showing the offered and used interfaces. The exact data was described in the separate interface specifications. This can be known as external context. Internal context existed and it was used inside the subsystems.

Context, both internal and external, has been distributed between subsystems but it has been used inside the base station. It is important to note that external context can be context that is dedicated either for the mobile phone user or for internal usage. The meaning of context that is going to, or coming from, the mobile phone user is meaningless for the base station but it needs memory to be processed. In pervasive computing, external context is always meaningful and dynamic. The difference is in the nature of context and the commonality is in the dynamicity of the context.

Recent research results into the pervasive computing state that:

- due to the inherent complexity of context-aware applications, development should be supported by adequate context-information modeling and reasoning techniques (Bettini et al., 2010)
- distributed context management, context-aware service modeling and engineering, context reasoning and quality of context, security and privacy, have not been well addressed in the Context-Aware Web Service Systems (Truong & Dustdar, 2009)
- development of context-aware applications is complex as there are many software engineering challenges stemming from the heterogeneity of context information sources, the imperfection of context information, and the necessity for reasoning on contextual situations that require application adaptations (Indulska & Nicklas, 2010)
- proper understanding of context and its relationship with adaptability is crucial in order to construct a new understanding for context-aware software development for pervasive computing environments (Soylu et al., 2009)
- ontology will play a crucial role in enabling the processing and sharing of information and knowledge of middleware (Hong et al., 2009)

3.1 Definitions

Many definitions for context as well for context-awareness are given in written research. The generic definition by Dey and Abowd for context and context-awareness are widely cited (Dey & Abowd, 1999):

'**Context** is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves. '

'**Context-awareness** is a property of a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. '

Context-awareness is also defined to mean that one is able to use context-information (Hong et al., 2009). Being context-aware will improve how software adapts to dynamic changes influenced by various factors during the operation of the software. Context-aware techniques have been widely applied in different types of applications, but still are limited to small-scale or single-organizational environments due to the lack of well-agreed interfaces, protocols, and models for exchanging context data (Truong & Dustdar, 2009).

In large embedded-software systems the user is not always the human being but can also be the other subsystem. Hence, the user has a wider meaning than in pervasive computing where the user, the human being, is in the center. We claim that pervasive computing will come closer to the user definition of embedded-software systems in the near future. Therefore, we propose that '*A context defines the limit of information usage of a smart space application*' (Toninelli et al., 2009). That is based on the assumption that any piece of data, at a given time, can be context for a given smart space application.

3.2 Designing the context

Concentrating on the context and changing the design from top-down to bottom-up while keeping the overall system in the mind is the solution to the challenges in the context-aware computing. Many approaches have been introduced for context modeling but we introduce one of the most cited classifications in (Strang & Linnhoff-Popien, 2004):

1. Key-Value Models

   The model of key-value pairs is the most simple data structure for modeling contextual information. The key-value pairs are easy to manage, but lack capabilities for sophisticated structuring for enabling efficient context retrieval algorithms.

2. Markup Scheme Models

   Common to all markup scheme modeling approaches is a hierarchical data structure consisting of markup tags with attributes and content. The content of the markup tags is usually recursively defined by other markup tags. Typical representatives of this kind of context modeling approach are profiles.

3. Graphical Model

   A very well-known general purpose modeling instrument is the UML which has a strong graphical component: UML diagrams. Due to its generic structure, UML is also appropriate to model the context.

4. Object-Oriented Models

   Common to object-oriented context modeling approaches is the intention to employ the main benefits of any object-oriented approach – namely encapsulation and reusability – to cover parts of the problems arising from the dynamics of the context in ubiquitous environments. The details of context processing are encapsulated on an object level and hence hidden to other components. Access to contextual information is provided through specified interfaces only.

5. Logic-Based Models

   A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from a set of other expressions or facts. To describe these conditions in a set of rules a formal system is applied. In a logic-based context model, the context is consequently defined as facts, expressions and rules. Usually contextual information is added to, updated in and deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. Common to all logic-based models is a high degree of formality.

6. Ontology-Based Models

   Ontologies are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers. Three ontology-based models are presented in this survey: i) Context Ontology Language (CoOL), (Strang et al., 2003); ii) the CONON context modeling approach (Wang et al., 2004); and iii) the CoBrA system (Chen et al., 2003a).

The survey of context modeling for pervasive cooperative learning covers the above-mentioned context modeling approaches and introduces a Machine Learning Modeling (MLM) approach that uses machine learning (ML) techniques. It concludes that to achieve the system design objectives, the use of ML approaches in combination with semantic context reasoning ontologies offers promising research directions to enable the effective implementation of context (Moore et al., 2007).

The role of ontologies has been emphasized in multitude of the surveys, e.g., (Baldauf et al., 2007), (Soylu et al., 2009), (Hong et al., 2009), (Truong & Dustdar, 2009). The survey related to context modeling and reasoning techniques (Bettini et al., 2010) highlights that ontological models of context provide clear advantages both in terms of heterogeneity and interoperability. Web Ontology Language, OWL, (OWL, 2004) is a de facto standard for describing context ontology. OWL is one of W3C recommendations (www.w3.org) for a Semantic Web. Graphical tools, such as Protégé and NeOnToolkit, exist for describing ontologies.

3.3 Context platform and storage

Eugster et al. present the middleware classification that they performed for 22 middleware platforms from the viewpoint of a developer of context-aware applications (Eugster et al., 2009). That is one of the many surveys done on the context-aware systems but it is interesting because of the developer viewpoint. They classified the platforms according to i) the type of context, ii) the given programming support, and iii) architectural dimensions such as decentralization, portability, and interoperability. The most relevant classification criteria of those are currently the high-level programming support and the three architectural dimensions.

High-level programming support means that the middleware platform adds a context storage and management. The three architectural dimensions are: (1) decentralization, (2) portability, and (3) interoperability. Decentralization measures a platform's dependence on specific components. Portability classifies platforms into two groups: portable platforms can run on many different operating systems, and operating system-dependent platforms, which can only run on few operating systems (usually one). Interoperability then measures the ease with which a platform can communicate with heterogeneous software components.

Ideal interoperable platforms can communicate with many different applications, regardless of the operating system on which they are built or of the programming language in which they are written. This kind of InterOperabilility Platform (IOP) is developed in the SOFIA-project (www.sofia-project.eu). The IOP's context storage is a Semantic Information Broker (SIB), which is a Resource Description Framework, RDF, (RDF, 2004) database. Software agents which are called Knowledge Processors (KP) can connect to the SIB and exchange information through an XML-based interaction protocol called Smart Space Access Protocol (SSAP). KPs use a Knowledge Processor Interface (KPI) to communicate with the SIB. KPs consume and produce RDF triples into the SIB according to the used ontology.

The IOP is proposed to be extended, where and when needed, with context-aware functionalities following 'the separation of concern' principle to keep application free of the context (Toninelli et al., 2009).

Kuusijärvi and Stenius illustrate how reusable KPs can be designed and implemented, i.e., how to apply 'for reuse' and 'with reuse' practices in the development of smart environments (Kuusijärvi & Stenius, 2011). Thus, they cover the need for programming level reusability.

3.4 Context-aware micro-architecture

When context information is described by OWL and ontologies, typically reasoning techniques will be based on a semantic approach, such as SPARQL Query Language for RDF (SPARQL), (Truong & Dustdar, 2009).

The context-awareness micro-architecture, CAMA, is the solution for managing adaptation based on context in smart environments. Context-awareness micro-architecture consists of three types of agents: context monitoring, context reasoning and context-based adaptation agents (Pantsar-Syväniemi et al., 2011a). These agents share information via the semantic database. Figure 4 illustrates the structural viewpoint of the logical context-awareness micro-architecture.

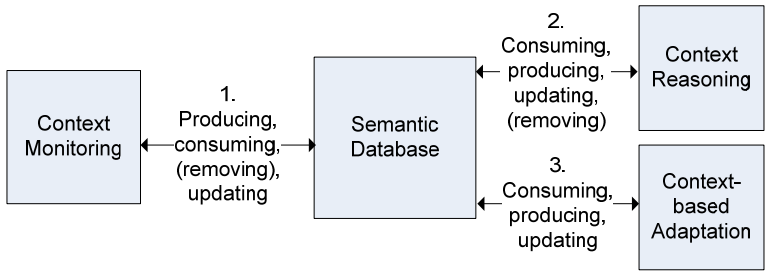

Fig. 4. The logical structure of the CAMA.

The context-monitoring agent is configured via configuration parameters which are defined by the architect of the intelligent application. The configuration parameters can be updated at run-time because the parameters follow the used context. The configuration parameters can be given by the ontology, i.e., a set of triples to match, or by a SPARQL query, if the monitored data is more complicated. The idea is that the context monitoring recognizes the current status of the context information and reports this to the semantic database. Later on, the reported information can be used in decision making.

The rule-based reasoning agent is based on a set of rules and a set of activation conditions for these rules. In practice, the rules are elaborated 'if-then-else' statements that drive activation of behaviors, i.e., activation patterns. The architect describes behavior by MSC diagrams with annotated behavior descriptions attached to the agents. Then, the behavior is transformed into SPARQL rules by the developer who exploits the MSC diagrams and the defined ontologies to create SPARQL queries. The developer also handles the dynamicity of the space by providing the means to change the rules at run-time. The context reasoning is a fully dynamic agent, whose actions are controlled by the dynamically changing rules (at run-time).

If the amount of agents producing and consuming inferred information is small, the rules can be checked by hand during the development phase of testing. If an unknown amount of agents are executing an unknown amount of rules, it may lead to a situation where one rule affects another rule in an unwanted way. A usual case is that two agents try to change the state of an intelligent object at the same time resulting in an unwanted situation. Therefore, there should be an automated way of checking all the rules and determining possible problems prior to executing them. Some of these problems can be solved by bringing

priorities into the rules, so that a single agent can determine what rules to execute at a given time. This, of course, implies that only one agent has rules affecting certain intelligent objects.

CAMA has been used:

- to activate required functionality according to the rules and existing situation(s) (Pantsar-Syväniemi et al., 2011a)
- to map context and domain-specific ontologies in a smart maintenance scenario for a context-aware supervision feature (Pantsar-Syväniemi et al., 2011b)
- in run-time security management for monitoring situations (Evesti & Pantsar-Syväniemi, 2010)

The Context Ontology for Smart Spaces, (CO4SS), is meant to be used together with the CAMA. It has been developed because the existing context ontologies were already few years old and not generic enough (Pantsar-Syväniemi et al, 2012). The objective of the CO4SS is to support the evolution management of the smart space: all smart spaces and their applications 'understand' the common language defined by it. Thus, the context ontology is used as a foundational ontology to which application-specific or run-time quality management concepts are mapped.

4. Conclusion

The role of software in large embedded systems, like in base stations, has changed remarkably in the last three decades; software has become more dominant compared to the role of hardware. The progression of processors and compilers has prepared the way for reuse and software product lines by means of C language, especially in the area of DSP software. Context-aware systems have been researched for many years and the maturity of the results has been growing. A similar evolution has happened with the object-oriented engineering that comes to DSP software. Although the methods were mature, it took many years to gain proper processors and compilers that support coding with C language. This shows that without hardware support there is no room to start to use the new methods.

The current progress of hardware development regarding size, cost and energy consumption is speeding up the appearance of context-aware systems. This necessitates that the information be distributed to our daily environment along with smart but separated things like sensors. The cooperation of the smart things by themselves and with human beings demands new kinds of embedded software. The new software is to be designed by the ontological approach and instead of the process being top-down, it should use the bottom-up way. The bottom-up way means that the smart space applications are formed from the small functionalities, micro-architecture, which can be configured at design time, on instantiation time and during run-time.

The new solution to designing the context management of context-aware systems from the bottom-up is context-aware micro-architecture, CAMA, which is meant to be used with CO4SS ontology. The CO4SS provides generic concepts of the smart spaces and is a common 'language'. The ontologies can be compared to the message-based interface specifications in the base stations. This solution can be the grounds for new initiatives or a body to start forming the 'borders', i.e., the system architecture, for the context-aware ecosystem.

5. Acknowledgment

The author thanks Eila Ovaska from the VTT Technical Research Centre and Olli Silvén from the University of Oulu for their valuable feedback.

6. References

Achillelos, A.; Yang, K. & Georgalas, N. (2009). Context modelling and a context-aware framework for pervasive service creation: A model-driven approach, *Pervasive and Mobile Computing*, Vol.6, No.2, (April, 2010), pp. 281-296, ISSN 1574-1192

Awad, M.; Kuusela, J. & Ziegler, J. (1996). *Object-Oriented Technology for Real-Time Systems. A Practical Approach Using OMT and Fusion*, Prentice-Hall Inc., ISBN 0-13-227943-6, Upper Saddle River, NJ, USA

Baldauf, M.; Dustdar, S. & Rosenberg, F. (2007). A survey on context-aware systems, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol.2, No.4., (June, 2007), pp. 263-277, ISSN 1743-8225

Bass, L.; Clements, P. & Kazman, R. (1998). *Software Architecture in Practice*, first ed., Addison-Wesley, ISBN 0-201-19930-0, Boston, MA, USA

Bettini, C.; Brdiczka, O.; Henricksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A. & Riboni D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Vol.6, No.2, (April, 2010), pp.161—180, ISSN 1574-1192

Bosch, J. (1999). Product-line architectures in industry: A case study, *Proceedings of ICSE 1999 21st International Conference on Software Engineering*, pp. 544-554, ISBN 1-58113-074-0, Los Angeles, CA, USA, May 16-22, 1999

Bosch, J. (2000). *Design and Use of Software Architectures. Adopting and evolving a product-line approach*, Addison-Wesley, ISBN 0-201-67484-7, Boston, MA, USA

Chen, H.; Finin, T. & Joshi, A. (2003a). Using OWL in a Pervasive Computing Broker, *Proceedings of AAMAS 2003 Workshop on Ontologies in Open Agent Systems*, pp.9-16, ISBN 1-58113-683-8, ACM, July, 2003

Clements, P.C.; Bachmann, F.; Bass L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R. & Stafford, J. (2003). *Documenting Software Architectures, Views and Beyond*, Addison-Wesley, ISBN 0-201-70372-6, Boston, MA, USA

Coleman, D.; Arnold, P.; Bodoff, S.; Dollin, C.; Gilchrist, H.; Hayes, F. & Jeremaes, P. (1993). *Object-Oriented Development – The Fusion Method*, Prentice Hall, ISBN 0-13-338823-9, Englewood Cliffs, NJ, USA

CPRI. (2003). Common Public Radio Interface, 9.10.2011, Available from http://www.cpri.info/

Dey, A. K. & Abowd, G. D. (1999). *Towards a Better Understanding of Context and Context-Awareness*. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, USA

Enders, A. & Rombach, D. (2003). *A Handbook of Software and Systems Engineering, Empirical Observations, Laws and Theories*, Pearson Education, ISBN 0-32-115420-7, Harlow, Essex, England, UK

Eugster, P. Th.; Garbinato, B. & Holzer, A. (2009) Middleware Support for Context-aware Applications. In: *Middleware for Network Eccentric and Mobile Applications* Garbinato, B.; Miranda, H. & Rodrigues, L. (eds.), pp. 305-322, Springer-Verlag, ISBN 978-3-642-10053-6, Berlin Heidelberg, Germany

Evesti, A. & Pantsar-Syväniemi, S. (2010). Towards micro architecture for security adaption, Proceedings of ECSA 2010 4th European Conference on Software Architecture Doctoral Symposium, Industrial Track and Workshops, pp. 181-188, Copenhagen, Denmark, August 23-26, 2010

France, R. & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. Proceedings of FOSE'07 International Conference on Future of Software Engineering, pp. 37-54, ISBN 0-7695-2829-5, IEEE Computer Society, Washington DC, USA, March, 2007

Goossens, G.; Van Praet, J.; Lanneer, D.; Geurts, W.; Kifli, A.; Liem, C. & Paulin, P. (1997) Embedded Software in Real-Time Signal Processing Systems: Design Technologies. *Proceedings of the IEEE*, Vol. 85, No.3, (March, 1997), pp.436–454, ISSN 0018-9219

Hillebrand, F. (1999). The Status and Development of the GSM Specifications, In: *GSM Evolutions Towards 3rd Generation Systems,* Zvonar, Z.; Jung, P. & Kammerlander, K., pp. 1-14, Kluwer Academic Publishers, ISBN 0-792-38351-6, Boston, USA

Hong, J.; Suh, E. & Kim, S. (2009). Context-aware systems: A literature review and classification. *Expert System with Applications*, Vol.36, No.4, (May 2009), pp. 8509-8522, ISSN 0957-4174

Indulska, J. & Nicklas, D. (2010). Introduction to the special issue on context modelling, reasoning and management, *Pervasive and Mobile Computing*, Vol.6, No.2, (April 2010), pp. 159-160, ISSN 1574-1192

Jacobson, I., et al. (1992). *Object-Oriented Software Engineering – A Use Case Driven Approach*, Addison-Wesley, ISBN 0-201-54435-0, Reading, MA, USA

Karlsson, E-A. (1995). *Software Reuse. A Holistic Approach*, Wiley, ISBN 0-471-95819-0, Chichester, UK

Kapitsaki, G. M.; Prezerakos, G. N.; Tselikas, N. D. & Venieris, I. S. (2009). Context-aware service engineering: A survey, *The Journal of Systems and Software*, Vol.82, No.8, (August, 2009), pp.1285-1297, ISSN 0164-1212

Kronlöf, K. (1993). *Method Integration: Concepts and Case Studies*, John Wiley & Sons, ISBN 0-471-93555-7, New York, USA

Krüchten, P. (1995). Architectural Blueprints—The "4+1" View Model of Software Architecture, *IEEE Software,* Vol.12, No.6, (November, 1995), pp.42-50, ISSN 0740-7459

Kuusijärvi, J. & Stenudd, S. (2011). Developing Reusable Knowledge Processors for Smart Environments, *Proceedings of SISS 2011 The Second International Workshop on "Semantic Interoperability for Smart Spaces" on 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011),* pp. 286-291, Munich, Germany, July 20, 2011

Miller J. & Mukerji, J. (2003). MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf

Moore, P.; Hu, B.; Zhu, X.; Campbell, W. & Ratcliffe, M. (2007). A Survey of Context Modeling for Pervasive Cooperative Learning, *Proceedings of the ISITAE'07 1st IEEE International Symposium on Information Technologies and Applications in Education*, pp.K51-K56, ISBN 978-1-4244-1385-0, Nov 23-25, 2007

Nokia Siemens Networks. (2011). Liquid Radio - Let traffic waves flow most efficiently. White paper. 17.11.2011, Available from http://www.nokiasiemensnetworks.com/portfolio/liquidnet

OBSAI. (2002). Open Base Station Architecture Initiative, 10.10.2011, Available from http://www.obsai.org/

OWL. (2004). Web Ontology Language Overview, W3C Recommendation, 29.11.2011, Available from http://www.w3.org/TR/owl-features/

Palmberg, C. & Martikainen, O. (2003) *Overcoming a Technological Discontinuity - The case of the Finnish telecom industry and the GSM*, Discussion Papers No.855, The Research Institute of the Finnish Economy, ETLA, Helsinki, Finland, ISSN 0781-6847

Pantsar-Syväniemi, S.; Taramaa, J. & Niemelä, E. (2006). Organizational evolution of digital signal processing software development, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol.18, No.4, (July/August, 2006), pp. 293-305, ISSN 1532-0618

Pantsar-Syväniemi, S. & Ovaska, E. (2010). Model based architecting with MARTE and SysML profiles. *Proceedings of SE 2010 IASTED International Conference on Software Engineering*, 677-013, Innsbruck, Austria, Feb 16-18, 2010

Pantsar-Syväniemi, S.; Kuusijärvi, J. & Ovaska, E. (2011a) Context-Awareness Micro-Architecture for Smart Spaces, *Proceedings of GPC 2011 6th International Conference on Grid and Pervasive Computing*, pp. 148–157, ISBN 978-3-642-20753-2, LNCS 6646, Oulu, Finland, May 11-13, 2011

Pantsar-Syväniemi, S.; Ovaska, E.; Ferrari, S.; Salmon Cinotti, T.; Zamagni, G.; Roffia, L.; Mattarozzi, S. & Nannini, V. (2011b) Case study: Context-aware supervision of a smart maintenance process, *Proceedings of SISS 2011 The Second International Workshop on "Semantic Interoperability for Smart Spaces", on 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011),* pp.309-314, Munich, Germany, July 20, 2011

Pantsar-Syväniemi, S.; Kuusijärvi, J. & Ovaska, E. (2012) Supporting Situation-Awareness in Smart Spaces, *Proceedings of GPC 2011 6th International Conference on Grid and Pervasive Computing Workshops*, pp. 14–23, ISBN 978-3-642-27915-7, LNCS 7096, Oulu, Finland, May 11, 2011

Paulin, P.G.; Liem, C.; Cornero, M.; Nacabal, F. & Goossens, G. (1997). Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends, *Proceedings of the IEEE,* Vol.85, No.3, (March, 2007), pp.419-435, ISSN 0018-9219

Pohl, K.; Böckle, G. & van der Linden, F. (2005). *Software Product Line Engineering*, Springer-Verlag, ISBN 3-540-24372-0, Berlin Heidelberg

Purhonen, A. (2002). *Quality Driven Multimode DSP Software Architecture Development*, VTT Electronics, ISBN 951-38-6005-1, Espoo, Finland

RDF. Resource Description Framework, 29.11.2011, Available from http://www.w3.org/RDF/

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. & Lorensen, W. (1991) *Object-Oriented Modeling and Design*, Prentice-Hall Inc., ISBN 0-13-629841-9, Upper Saddle River, NJ, USA

Shaw, M. (1990). Toward High-Level Abstraction for Software Systems, *Data and Knowledge Engineering,* Vol. 5, No.2, (July 1990), pp. 119-128, ISSN 0169-023X

Shlaer, S. & Mellor, S.J. (1992) *Object Lifecycles: Modeling the World in States*, Prentice-Hall, ISBN 0-13-629940-7, Upper Saddle River, NJ, USA

Soylu, A.; De Causmaecker1, P. & Desmet, P. (2009). Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological

Engineering, *Journal of Software*, Vol.4, No.9, (November, 2009), pp.992-1013, ISSN 1796-217X

SPARQL. SPARQL Query Language for RDF, W3C Recommendation, 29.11.2011, Available from http://www.w3.org/TR/rdf-sparql-query/

Strang, T.; Linnhoff-Popien, C. & Frank, K. (2003). CoOL: A Context Ontology Language to enable Contextual Interoperability, *Proceedings of DAIS2003 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, pp.236-247, LNCS 2893, Springer-Verlag, ISBN 978-3-540-20529-6, Paris, France, November 18-21, 2003

Strang, T. & Linnhoff-Popien, C. (2004). A context modelling survey, *Proceedings of UbiComp 2004 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pp.31-41, Nottingham, England, September, 2004

Toninelli, A.; Pantsar-Syväniemi, S.; Bellavista, P. & Ovaska, E. (2009) Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability, *Proceedings of M-PAC'09 International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, session: short papers, Article No: 5, ISBN 978-1-60558-849-0, Urbana Champaign, Illinois, USA, November 30, 2009

Truong, H. & Dustdar, S. (2009). A Survey on Context-aware Web Service Systems. *International Journal of Web Information Systems*, Vol.5, No.1, pp. 5-31, ISSN 1744-0084

Wang, X. H.; Zhang, D. Q.; Gu, T. & Pung, H. K. (2004). Ontology Based Context Modeling and Reasoning using OWL, *Proceedings of PerComW '04 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 18–22, ISBN 0-7695-2106-1, Orlando, Florida, USA, March 14-17, 2004

PUBLICATION II

Organizational evolution of digital signal processing software development

In: Journal of Software Maintenance and Evolution: Research and Practice 2006(18), pp. 293–305.

JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.334

Practice

Organizational evolution of digital signal processing software development

Susanna Pantsar-Syväniemi[1,*,†], Jorma Taramaa[1] and Eila Niemelä[2]

[1]*Nokia Networks, P.O. Box 319, FIN-90651 Oulu, Finland*
[2]*VTT Technical Research Centre of Finland, P.O. Box 1100, FIN-90571 Oulu, Finland*

SUMMARY

A base station, as a network element, has become an increasingly software-intensive system. Digital signal processing (DSP) software is hard real-time software that is a part of the software system needed in a base station. This article reports practical experiences related to organizing the development of embedded software in the telecommunication industry, at Nokia Networks. The article introduces the main factors influencing the development of DSP software and also compares the evolutionary process under study with both selected organizational models for a software product line and a multistage model for the software life cycle. We believe it is vitally important to formulate the organization according to the software architecture, and it is essential to have a dedicated development organization with long-term responsibility for the software. History shows that without long-term responsibility, there is no software reuse. In this paper we introduce a new organizational model for product line development. This new hybrid model clarifies long-term responsibilities in large software organizations with hundreds of staff members and formulates the organization according to the software architecture. Our case needs a couple more constraints to keep it in the evolution stage of the software life cycle. Thus, we extend the evolution phase in the multistage model to make it relevant for embedded, hard real-time software. Copyright © 2006 John Wiley & Sons, Ltd.

Received 15 November 2004; Revised 19 February 2006; Accepted 20 April 2006

KEY WORDS: software product line; software reuse; embedded software; software architecture; software maintenance; software development

1. INTRODUCTION

How can we shorten a company's time-to-market? How can we widen its product portfolio? How can we improve the quality and performance of the company's products? These questions are essential

*Correspondence to: Susanna Pantsar-Syväniemi, Nokia Networks, P.O. Box 319, FIN-90651 Oulu, Finland.
†E-mail: susanna.pantsar-syvaniemi@nokia.com

in today's telecommunication business, and they drive a need to rein in the efforts expended in software development in order to meet business demands. The natural way to decrease these efforts is to increase reuse. Software reuse based on product line architecture is successfully applied in the software industry [1]. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of reusable core assets in a prescribed way [2]. Software architecture is used as a means of abstraction, communication and management [3], and therefore it is the key asset in a product line organization. Software changes are always present [4,5]; a system that is used will be changed, and its complexity will increase unless work is done to reduce it. Changes initiated by the use of architecture generally affect the entire organization [3]. Therefore, full acceptance of organizational changes is important in making successful use of architecture. Managing changes through architecture has many similarities with other evolutionary processes [4,5]. In [6], software architecture is considered a main research topic in handling unanticipated changes, controlling software evolution and managing the knowledge and expertise of software teams. Moreover, raising the level of abstraction is seen as a promising means of managing software evolution.

This article describes experiences with organizational evolution in the area of digital signal processing (DSP) software as a part of base station development at Nokia Networks, and it proposes a new organizational model for a DSP software product line as well as the whole base station software product line. The findings are based on experiences gained while the authors worked in different areas of embedded software within projects and line management in base station software development. The findings are compared with organizational alternatives for software product lines introduced in [7]. We have chosen these alternatives because they seemed to be the most suitable for our case and their roots are within industry. The responsibility for software maintenance of a base station product is significant, lasting longer than 10 years. Owing to this long-lasting nature of maintenance, we compare our case (DSP as embedded, hard real-time software) with the multistage model for the software life cycle introduced in [6].

This paper is composed as follows. The next section introduces the factors influencing the development of DSP software within base station software. Section 3 describes the characteristics of both the chosen organizational alternatives when applying the software product line approach, and the multistage model for the software life cycle. Section 4 presents the organizational evolution of DSP software development at Nokia Networks. Section 5 discusses the results of the comparison and introduces both the new organizational model for a software product line and the extensions of the evolution phase in the multistage model. The final remarks are presented and future research needs are discussed in Section 6.

2. FACTORS INFLUENCING BASE STATION DSP SOFTWARE

DSP software operates with very tight timing, memory and performance requirements. The DSP is a central part of the physical layer base band solutions of telecommunications (or mobile wireless) systems, such as digital mobile phones and digital base stations. In general, the functions of the physical layer have been implemented in hardware, e.g., ASIC (application-specific integrated circuit), FPGA (field programmable gate array) or near to hardware [8,9].

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

Traditionally, systematic reuse has been difficult in DSP software because each product has required careful optimization of resources, and the implementation language has been mainly a processor-specific assembly. A software architecture creates a common basis between software development and systematic software reuse. It can be used to form a common understanding of what is being developed between different stakeholders, such as software designers, architects, managers, and marketing people [10].

As processors and compilers have become more efficient, ever since the introduction of WCDMA (Wideband Code-Division Multiple Access) air interface technology in UMTS (Universal Mobile Telecommunication System), the C language has been used instead of assembly in DSP software development. The use of C is one of the enabling factors on the way to making reusable DSP software [10]. Another enabling factor is more advanced tools, making it possible to separate DSP software development from the underlying platform.

Nowadays, base station software consists of a huge number of different types of embedded software, i.e., real-time control and management, hardware-related and hard real-time software. Hard real-time software comprises DSP software. User interface software is developed along with embedded base station software. Each type of software has its own characteristics. For example, DSP software has an 'engagement' with ASICs and FPGAs, and time-critical aspects are connected to the source code implementation. The base station business model has changed from a closed, in-house business to distributed development on an open market, bringing new stakeholders on the scene. An open market requires a change towards a modular, service-oriented architecture that influences the manner of communication. The need to distribute information among geographically separated and/or culturally different development teams is solved by defining the stakeholders of the domain and introducing the viewpoints necessary for service-oriented module development [11].

Standards have a major role in the telecommunication industry. They define many facts via specifications, like communication between different parties. During the 1980s, European telecommunication organizations and companies reached a common understanding on the development of a Pan-European mobile communication standard, Global System for Mobile Communications (GSM), by establishing a dedicated organization, ETSI (European Telecommunications Standards Institute), for the further evolvement of the GSM air interface standard. This organization has produced the GSM 900 and 1800 standard specifications [12]. The large European telecommunication companies created a consortium. The goal of this group was to prepare a new European mobile telephone network, GSM [13]. Nokia participated in this consortium, being one of the developers of a novel GSM digital base station solution.

Development of the GSM standard included more and more challenging features of standard mobile technology defined by ETSI, such as HSCSD (High Speed Circuit Switched Data), GPRS (General Packet Radio Service), AMR (Adaptive Multirate Codec) and EDGE (Enhanced Data rates for GSM Evolution) [12]. One should recognize the fact that UMTS (Universal Mobile Telecommunication System) essentially should be interpreted as a continuation of the regulatory regime and technological path set in motion through GSM, rather than a radical break from this regime. In effect, GSM standardization defined a path of progress through GPRS and EDGE towards UMTS as the major standard of 3G under the 3GPP standardization organization [14].

3GPP within ETSI [15] is a body that serves the players who benefit the most from network and element-level standardization, i.e., network suppliers and network operators. However, these bodies do not standardize the internal interfaces of network elements. For this purpose we need industry forums

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

such as OBSAI (Open Base Station Architecture Initiative). OBSAI was set up to define and agree on open standards for base station internal architecture and key interfaces. Module vendors can develop modules with the specified interface and sell them to base station manufacturers [16].

From the above we can highlight that the issues that set the main constraints for organizing initial development and maintenance of DSP software are the underlying hardware, standards, and a common architecture. It is essential to note that these constraints have different lifetimes. Hardware evolves according to 'Moore's Law' [5], and that progress is much more rapid than the evolution of ETSI [15] and OBSAI [16] standards.

3. FRAMES OF COMPARISON

As stated earlier, we selected two frames of comparison: organizational alternatives for software product lines and a multistage model for the software life cycle. To be able to foresee and deal with the factors introduced in Section 2, the former frame provides an appropriate software product line for our case, which is embedded, hard real-time software and a part of a huge embedded software system needed for a base station as a network element. Another aspect of the former frame is that the most suitable organizational alternative has to be changed according to varying needs in the company's software development. It is interesting to see how this aspect correlates with our case. The latter frame gives a deeper view of software maintenance and evolution in our case.

3.1. Organizational alternatives for software product lines

The strength of the software product line is that it clarifies responsibility issues in creating, modifying and maintaining the software needed for the company's products. The characteristics of organizational alternatives are grouped around four main models: the development department, business units, the domain engineering unit and the hierarchical domain engineering unit [7]. These models are described in Table I.

3.2. Multistage model for the software life cycle

The multistage model presents the software life cycle as a sequence of stages, with initial development being the first stage [6]. It separates the 'maintenance' phase into an evolution stage followed by servicing and phase-out stages, as shown in Figure 1. A few characteristics of the stages are listed in Table II.

4. CHARACTERISTICS OF BASE STATION DSP SOFTWARE DEVELOPMENT

As mentioned earlier, our focus is on DSP software within base station software. Base station systems in the early 1990s were hardware based, and software's task was mainly to control the hardware. A big achievement of that time was that engineers cooperated with the GSM standardization body. Development work was done within one site, meaning the software engineers were gathered together to work side by side. Thus, information sharing within the development department was handled easily and quickly. When base station software development was in its initial phase, there were only a few

DOI: 10.1002/smr

Table I. Characteristics of organizational alternatives for software product lines.

Organizational alternatives	Number of software staff	Responsibility for assets	How work is organized
Development department	<30	A single organizational unit that develops and maintains reusable assets	No permanent structure, work is typically organized in projects
Business units	30–100	Each business unit is responsible for the development and evolution of one product or a few products Business units share reusable assets	Product (type of systems) centric
Domain engineering unit	100–hundreds	Separates the development and evolution of shared assets from the development of concrete systems One (or multiple) domain engineering unit(s) and several product engineering units	The domain engineering unit distributes software architecture and components to the product (system) engineering units
Hierarchical domain engineering units	Hundreds	Specialized domain engineering units that develop and evolve reusable assets for a subset of the products in the family	Architecture-centric: general assets for all, specialized assets for a set of products

software engineers working in DSP software development at the time. The physical layer solution was implemented using both hardware and DSP software. Development was managed by hardware technologies. More characteristics of software development of that time are described in Table III.

4.1. In the mid-1990s

Even though base station development was hardware oriented, the role of DSP software was growing in the physical layer implementation. GSM recommendations required flexible control of hardware, which could only be achieved by using DSP software. In practice, this still meant, however, that hardware persons made the decisions of codesign. The DSP software made it possible to correct the functionality of hardware and to evolve old base stations with novel physical layer features by producing new software. Software provided real business benefits.

The DSP software was coded with a processor-specific assembler language. Development of base station software was done on a project basis, and the project was organized according to the architecture that was familiar from the beginning. There were just a few software engineers specializing in DSP software. Roughly, the size of the whole base station software department was in the order of tens of employees. The characteristics of software development at that time are described in Table IV.

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

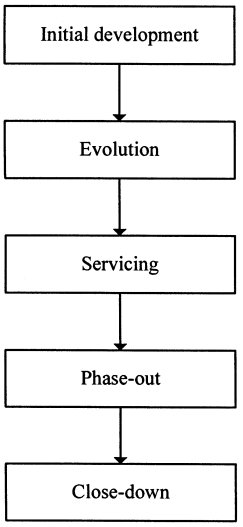

Figure 1. Multistage model.

After the first version of the whole base station software was ready to be delivered to customers, the next version was under development in a new project, based on the version created earlier. The new project took care of developing new features for the base station and also maintaining the software made earlier.

4.2. At the end of the 1990s

'Moore's Law' [5] and the evolving ETSI standards [15] were the primary factors behind the need to scale development work. The natural way to scale was to establish new development projects for different products and separate projects for software maintenance. In DSP software this meant the newer base station was going to have a newer processor than the earlier base station product(s). In addition, a typical trend of that time was an expansion of the number of software developers. This situation forced us to build a multisite organization.

The most essential nature of DSP development at that time was adding new GSM features into the existing base station hardware platform with DSP software. This extended the application knowledge of DSP software developers, but similar features were developed in parallel projects by different DSP software developers. Still, software development was organized according to the architecture. There were no other assets than the common architecture. The projects had the resources to create their own solutions even for common tasks or features. The characteristics of this development phase are introduced in Table V.

DOI: 10.1002/smr

Table II. Characteristics of the multistage model for the software life cycle.

Stage	Characteristics of the stage
Initial development	The first version of the software system is developed. This stage provides knowledge about the application domain, user requirements, the role of the application in the business process, the operating environment, program architecture, etc.
Evolution	After successful initial development, the software is in the evolution stage. The goal is to adapt the application to the ever-changing user requirements and operating environment. It corrects faults in the application. The software is being evolved because it is successful in the marketplace (revenue streams are buoyant, user demand is strong, the development atmosphere is vibrant and positive, the organization is supportive). Return on investment is excellent. Both the software architecture and the knowledge of the software team make evolution possible
Servicing	Only minor tactical changes (patches, code changes, wrappers) are possible. They further deteriorate the architecture. The software is likely to no longer be a core product for business, the cost benefits of changes are much more marginal. If the knowledge necessary for evolution is lost, changes in the software will lead to faster deterioration of the software architecture
Phase-out	No more servicing is being undertaken, but the system may still be in production
Close-down	Software use is disconnected and the users are directed towards a replacement

Table III. Characteristics of software development in the beginning.

Number of software persons in the base station software department	Number of software persons developing DSP software	Language used in DSP software	Products	Development work	Organizational units
<20	<5	Processor-specific assembler	GSM base station	Hardware-oriented product development The physical layer was implemented using both hardware and DSP software	Project according to the product under development

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

Table IV. Characteristics of software development in the mid-1990s.

Number of software persons in the base station software department	Number of software persons developing DSP software	Language used in DSP software	Products	Development work	Organizational units
30–60	<10	Processor-specific assembler	GSM base station	Hardware-oriented product development DSP software achieved a significant role in the physical layer	Project; no permanent unit for handling software maintenance Software development was organized according to the architecture The common architecture was an asset

Table V. Characteristics of software development at the end of the 1990s.

Number of software persons in the base station software department	Number of software persons developing DSP software	Language used in DSP software	Products	Development work	Organizational units
<200	<50	Processor-specific assembler	GSM base stations	Project specific domain engineering	Concurrent multi-site projects

4.3. In the early 2000s

After the concurrent development teams in different projects for base station products, the next major step in the organizational evolution was the establishment of reuse activities; development 'for reuse' and development 'with reuse'. 'For reuse' means development of reusable assets and 'with reuse' means using the assets in product development or maintenance [17]. The main problem within this process-centric, 'for reuse' and 'with reuse', development was that it produced an architecture that was too abstract. The reason was that the domain was too wide, i.e., the domain was base station software in its entirety. Project-specific solutions were created during reuse activities despite the fact

DOI: 10.1002/smr

Table VI. Characteristics of the software development in the early 2000s.

Number of software persons in the base station software department	Number of software persons developing DSP software	Language used in DSP software	Products	Development work	Organizational units
>200	>50	C	GSM and WCDMA base stations	Reuse-based	Architecture-centric

that software reuse was meant to happen. Thus, the meaning and benefits of domain engineering were not understood. The scope within one specific project was to get a certain base station product ready for the market.

Usually, one project will carry the responsibility for developing a specific and new base station product. Afterwards, this project does not have long-term responsibility for maintaining the product or the software embedded in the product. In effect, product concepts and software development will change quite a bit. As a consequence of this change there will be a need to assign responsibility for development and maintenance of the software to a certain organizational unit. Owing to the temporal nature of a project, maintenance should be assigned to a 'permanent' unit that is not a separate project. The characteristics of software development in this period are described in Table VI. The commitment of the line and project management is vital for domain engineering. The aim to create a foundation for software reuse can be quite easily sacrificed because of business pressure or unsuitable responsibility issues between line organizations and project(s).

4.4. In the mid-2000s

Here the organizational unit's responsibility means that the unit, such as the DSP software development department, has long-term responsibility for different specifications, interfaces and software modules. This situation in DSP software development has been analyzed in [18]. The unit is also responsible for the software architecture in its own domain. Domain architectures are defined according to the architecture of the whole base station system.

As a very large software development organization with hundreds of developers, the base station software development department is too large to be a pure domain engineering unit, and because of the open module business, a new organizational model for software development is needed. Adoption of a product line architecture is required both in module service development and in the development of base station services [11]. Domain development is not a separate activity to product development. On the contrary, domain engineering has to be closely linked to the development of products; even the development of new features is primarily part of the activities in application (or system or product) engineering.

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

5. RESULTS

In this section the findings based on experiences and the comparison between the case under study and the chosen frames are discussed. The purpose is to identify both a suitable software product line and correlations between the evolutionary path and the organizational alternatives for software product lines introduced earlier in this article. Another idea is to clarify our software maintenance and evolution against the frame for the software life cycle.

5.1. New organizational alternative for a software product line

During the evolutionary path a project-based style was used to organize the software engineers and the development of base stations was hardware oriented. Establishing new projects for different base station products scaled development. The fundamental issue was the growing number of projects within the evolving technologies of the telecommunication infrastructure industry, where maintenance has to be provided for sold products for at least 10 years. The maintenance of software was organized on the basis of a product or product family.

In the project-based phase within our evolutionary process the software engineers were assigned to teams according to the common architecture for the duration of a project. The common software architecture was good and relevant for development at that time, but it was no longer suitable when the diversity of products increased. As mentioned earlier, our development department was working like a product line. In the product-based phase, we also used a common architecture. In this phase we scaled development by establishing new product projects within a product line and by establishing product lines for separate business units. A characteristic of this approach was the expansion of the number of software developers. That situation forced us to build a multisite organization. In the 'for reuse' and 'with reuse' phase, we tended to build up a platform for sharing components and architecture. This phase was a natural step in the evolutionary path towards the new model needed for service-oriented software development.

Our solution for a new model is a hybrid model, which is a combination of the hierarchical domain model for the base station product line and the development departments for module (such as baseband) development. This hybrid model is a newcomer in the field of organizing software development in very large organizations. The difference between our hybrid model and the hierarchical domain engineering model [7] is that we prefer a development department to a domain engineering unit. We consider that, for the module product line, the development department would be more useful than the pure domain engineering unit because of clear responsibility sharing. The development department type of product line has overall responsibility for domain and application engineering projects as well as responsibility for the common software architecture of all products to be produced within this product line. Based on our experiences, our evolutionary path in the context of organizational units (as shown in Table VII) has evolved from project type to product type, continuing via the platform-centric mode to the architecture-centric mode for organizing the developers.

With the hybrid model, we will be able to have the long-term responsibility that is the key issue in the context of software reuse. Herein software reuse is based on product line architectures. The motivation for using the hybrid model is to be able to produce large, growing and complicated software efficiently and cost-effectively. With a 'permanent unit' and clear responsibilities, the organizations will be able to achieve an easier continuum for ever-evolving hardware platforms and development tools independent from the hardware platform.

DOI: 10.1002/smr

Table VII. Similarities and differences between the case under study and organizational alternatives for a software product line.

Case under study	Correlating organizational alternative according to [7]	Assets in base station software	The way development work was organized within the evolutionary step in the case
Starting phase	Development department	Common architecture	Project
Concurrent development groups	Development departments in separate business units	Common architecture	Product
'For reuse' and 'with reuse' development projects	Domain engineering unit	Platform, shared components	Process-centric
Service-oriented module development (hybrid model)	Combination of the development department and the hierarchical domain engineering units	Shared components according to the architecture	Architecture-centric

In the sense of DSP software, this means the DSP software developers are responsible for development and maintenance of the assets needed in the DSP software area. When the responsibility is divided according to the architecture, it is also good to organize the software developers accordingly. This can be called a competence-centric way of organizing the software developers. We consider the architecture or competence-centric way to be the best choice for organizing the staff for a system like the base station, with long-term maintenance responsibility for the customers.

5.2. Extended staged model for software life cycle

Based on our experience, we see that DSP software inside the base station needs a few more constraints for the evolution stage in the software life cycle because of very tight timing requirements and a close relationship between hardware and DSP software. DSP software is in the evolution stage as long as it can be optimized or the underlying hardware is capable enough. At some point, financial reasons will lead the software to the servicing stage.

6. FINAL REMARKS

In this paper we introduced a company's evolutionary process for organizing DSP software development. The amount of software has grown during the years. The role of software has also changed: it has become more dominant. The underlying hardware, standards and a common architecture were the issues that set the main constraints for organizing development and maintenance

DOI: 10.1002/smr

of the software. As the size of the company and the number of projects and products increased, it became necessary to reorganize the structure of working. A development department with a common architecture was applied when only a few software developers were involved. In concurrent software engineering, software development and maintenance of the software were allocated to separate business units. When the focus was on a platform and shared components, software development was process-centric: 'for reuse' and 'with reuse' development. Shifting towards service-orientation required a new organization model: a hybrid model, which is a combination of the development departments and the hierarchical domain engineering units.

In [4], it was argued that, to a certain extent, the evolution phenomenon controls the organization. We hold the view that the given argument is true when it comes to maintenance and evolution of the software. In the initial development phase of the software, it might be an organization that leads the evolution by influencing customers to make a certain evolution happen. As far as the multistage model is concerned, we found the need to expand its evolution stage to be relevant for our case as embedded and hard real-time software. The expansions were to make the underlying hardware efficient enough and to optimize the software itself.

To successfully reuse software, the commitment to reuse has to exist at every level in the organization. Communication is most essential for getting this commitment. Software architecture is really important in sharing the information and the reasons; architecture is vital for reuse. The aim of creating a foundation for software reuse can quite easily be sacrificed because of business pressure or unclear responsibility for sharing between the line organization and project(s).

Work will continue within the hybrid model introduced for the first time in this paper. Upcoming research should also consider how to progress from a product program-focused mode to reuse-driven software development.

REFERENCES

1. Bosch J. Product-line architectures in industry: A case study. *Proceedings 21st International Conference on Software Engineering (ICSE'99)*. IEEE Computer Society Press: Los Alamitos CA, 1999; 544–554.
2. Clements P, Bachman F, Bass L, Garland D, Ivers J, Little R, Nord R, Stafford J. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley: Boston MA, 2003; 294 pp.
3. van der Raadt B, Soetendal J, Perdeck M, van Vliet H. Polyphony in architecture. *Proceedings 26th International Conference on Software Engineering (ICSE 2004)*. IEEE Computer Society Press: Washington DC, 2004; 533–542.
4. Lehman MM, Belady LA. *Program Evolution: Processes of Software Change*. Academic Press: London, 1985; 1–38, 275–287.
5. Enders A, Rombach D. *A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories*. Pearson Education: Harlow, 2003; 160–177, 244.
6. Bennett KH, Rajlich VT. Software maintenance and evolution: A roadmap. *Proceedings of the Conference on the Future of Software Engineering*. ACM Press: New York NY, 2000; 73–87.
7. Bosch J. Software product lines: Organizational alternatives. *Proceedings 23rd International Conference on Software Engineering (ICSE 2001)*. IEEE Computer Society Press: Washington DC, 2001; 91–100.
8. Paulin PG, Liem C, Cornero M, Nacabal F, Goossens G. Embedded software in real-time signal processing systems: Application and architecture trends. *Proceedings of the IEEE* 1997; **85**(3):419–434.
9. Goossens G, Van Praet J, Lanneer D, Geurts W, Kifli A, Liem C, Paulin P. Embedded software in real-time signal processing systems: Design technologies. *Proceedings of the IEEE* 1997; **85**(3):436–454.
10. Purhonen A. *Quality Driven Multimode DSP Software Architecture Development*. VTT Electronics: Espoo, Finland, 2002; 41–52.
11. Matinlassi M, Pantsar-Syväniemi S, Niemelä E. Towards service-oriented development in base station modules. *Proceedings of the 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004)*, vol. 2, Trappl R (ed.). Austrian Society for Cybernetic Studies: Vienna, 2004; 440–444.

DOI: 10.1002/smr

12. Hillebrand F. The status and development of the GSM specifications. *GSM Evolutions Towards 3rd Generation Systems*. Zvonar Z, Jung P, Kammerlander K (eds). Kluwer Academic Publishers: Dordrecht, 1999; 1–14.
13. Nokia. The history of Nokia 1865–2002. http://www.nokia.com/A402756 [19 February 2006].
14. Palmberg C, Martikainen O. Overcoming a technological discontinuity: The case of the Finnish telecom industry and the GSM. *Discussion Papers No.855*, The Research Institute of the Finnish Economy, Helsinki, Finland, 2003; 55 pp. Available at: http://www.etla.fi/files/677_dp855.pdf [10 February 2006].
15. ETSI. Welcome to ETSI. The European Telecommunications Standards Institute: Sophia Antipolis, France, 2004. http://www.etsi.org/home.html [14 February 2006].
16. OBSAI. Welcome to OBSAI. Open Base Station Architecture Initiative, 2002; 1 pp. http://www.obsai.org/ [14 February 2006].
17. Karlsson E-A. *Software Reuse. A Holistic Approach*. Wiley: Chichester, 1995; 3–34, 249–270, 287–376.
18. Kääriäinen J, Taramaa J, Alenius J. Configuration management support for the development of an embedded system: Experiences in the telecommunication industry. *Proceedings of the 5th International Symposium on Tools and Methods of Competitive Engineering (TMCE 2004)*, vol. 2. Millpress: Rotterdam, 2004; 605–616.

AUTHORS' BIOGRAPHIES

Susanna Pantsar-Syväniemi is a Software Specialist in WCDMA RAN Research and Development at Nokia Networks, Oulu, Finland. Since 1993, she has been working for embedded systems with Nokia Networks. Her research topic is related to organizational issues on company's reuse strategy. She graduated (MSc, in 1993) in Information Processing from the Lappeenranta University of Technology, Finland. She is a postgraduate student in information processing science at the University of Oulu, Finland.

Jorma Taramaa is a R&D Manager at Nokia Networks, Oulu, Finland with 20 year's experience in software research and development of embedded systems. He finished his doctoral thesis on software configuration management of embedded systems in 1998 at the University of Oulu, Finland. As a R&D Manager, he has been responsible for competence management of digital signal processing software for WCDMA-based basestations.

Eila Niemelä is a Research Professor at VTT Technical Research Centre of Finland. Her main research interests are product family architectures, service architectures, quality-driven architecture design and quality evaluation. Since 2002 she has also worked as a docent of software architectures and components at the University of Oulu. She obtained her MSc degree in 1995 and the PhD degree in 2000 in Information Processing Science from the University of Oulu, Finland.

J. Softw. Maint. Evol.: Res. Pract. 2006; **18**:293–305
DOI: 10.1002/smr

PUBLICATION III

Situation-based and self-adaptive applications for the smart environment

In: Journal of Ambient Intelligence and Smart Environments 2012 (4), pp. 491–516.

Journal of Ambient Intelligence and Smart Environments 4 (2012) 491–516
DOI 10.3233/AIS-2012-0179
IOS Press

Situation-based and self-adaptive applications for the smart environment

Susanna Pantsar-Syväniemi*, Anu Purhonen, Eila Ovaska, Jarkko Kuusijärvi and Antti Evesti
VTT Technical Research Centre of Finland, P.O. Box 1100, FI-90571 Oulu, Finland
E-mail: {susanna.pantsar-syvaniemi,anu.purhonen,eila.ovaska,jarkko.kuusijarvi,antti.evesti}@vtt.fi

Abstract. Situation-based and self-adaptive applications are the key enablers of smart environments and ecosystems. In those environments, developers and users focus on innovating and making added-value applications, instead of solving the problems of interoperability and complexity of heterogeneous systems. This paper contributes by introducing an innovative adaptation framework for the situation-based and self-adaptive applications of smart environments. The framework embodies a novel architecture, generic ontologies for context, security, and performance management, and dynamic models for performing run-time reasoning and adaptation. The framework is intended for an application developer who is i) creating application scenarios, and ii) transforming the scenarios into annotated sequence diagrams with the help of the static models of the framework, the ontologies, and the rules defined in them. Thereafter, the application developer iii) transforms the annotated application behavior description into the selected rule language, SPARQL. The approach is exemplified through the creation of the GuideMe application, which exploits context, security, and performance information to adapt the service according to the quality requirements and the context of the user, as well as the smart environment, without bothering the end-user.

Keywords: Run time, ontology, context-awareness, performance, security

1. Introduction

Recently, an increasing amount of research funding has been dedicated to developing technologies and applications for smart environments and smart spaces. A smart environment is a physical space where smart applications that are executed locally or globally interact in order to achieve intelligence that fulfills an emerging need of space users by answering the following questions: What is the space used for? Who is using the space? What roles, rights and responsibilities might users have? Thus, the smart space emphasizes the purpose of an environment, and the 'smart environment' is its physical realization formed from a set of physical things (i.e., devices, actuators, sensors, etc.) that are able to interoperate in an ambient way.

Because of the dynamicity of emerging and disappearing devices, the overall structure of a smart environment cannot be fully specified at design time. However, the co-operating software subsystems that create the overall system at any specific time need to be designed and tested.

The biggest obstacles in establishing smart environments are heterogeneity and complexity of computing, communication, and software technologies. These technologies more likely hinder rather than promote the construction of innovative applications based on evolving devices and systems and changing user and business requirements. Dynamism is at the core of smart environments, and therefore applications should be able to react and even anticipate changes that will happen inside and outside of the application execution platform [6]. Being reactive and proactive requires that the application be aware of the changes happening around it and that it be able to reason based on available information about the situation. The application is to be interoperable with other applications, the platform, users and their surroundings. Thus, solutions for interoperability are required at different levels, such as connectivity, data, the semantics of data, context, change of context, and application behavior [32].

The main contribution of this paper is a reusable adaptation framework for developing situation-based

* Corresponding author.

applications for smart environments. It is an extension to existing implementations of the interoperability platform (IOP) that do not support self-management features. The IOP is a solution for achieving interoperability of information between heterogeneous devices and systems in a smart space. The IOP has been developed within the SOFIA/ARTEMIS project[1]. Requirements of the IOP are summarized in the IOP principles, which define the style followed in architecting the IOP [32].

The main principles of the framework defined are simplicity, self-containment, and self-adaptiveness. Simplicity means that part of the framework complies with the architectural pattern, the MAPE-K (Monitor, Analyze, Plan, and Execute – Knowledge). The MAPE part of the pattern is realized with software agents. Self-containment means that each agent has a well-defined objective, semantics and up-to-date (meta) data necessary for achieving this objective. Thus, the K-part, or knowledge, is separated from the MAPE agents. Self-adaptiveness guarantees that the agent behaves as an intelligent entity as part of the application, and thereby realizes the smartness of the environments. The semantic models, or ontologies, guarantee interoperability beyond communication and interoperability of the information exchanged. They also guarantee interoperability of context and its changes, interoperability of application behavior and alignment of the concepts on which the smart environment is based.

The framework is claimed to be generic; it can be applied to any type of smart environment because it solves the main problems of most physical environments: semantic, dynamic, behavioral, and conceptual interoperability; and self-adaptiveness. The framework is also generic in the sense that it is independent of an application domain and architectural style; it can be applied in component-based and in service-based systems. The concept is open because the legacy issues that do not support situation-based behavior can operate with the things that have situation-based functionality. There is no fixed behavior in the situation-based and self-adaptive applications. The behavior emerges dynamically from the communication of intelligent things composed according to the behavioral model and the rules defined based on this model.

The main benefits of the proposed framework are: i) a reusable architecture that provides an execution platform for situation-based and self-adaptive applications; ii) reusable conceptual models for context, security, performance, and domain; and iii) a process for defining behavioral models for applications and configuring application behavior in the design and instantiation phases, as well as at run time.

The structure of the paper is as follows: The next section presents the related work. Section 3 introduces the analyses of requirements and solutions. Section 4 details the framework for self-adaptive and situation-based application agents and the semantic models they use for information processing. Section 5 introduces the process to develop an application with the adaptation framework and lessons learned. Section 6 discusses the evaluation process, the experiments, and the Seamless Usage of Multiple Smart Spaces (SUM-SS) pilot carried out in the SOFIA project. It also goes through the evolution management of smart spaces and future work. Conclusions close the paper.

2. Related research

2.1. Adaptation frameworks

The defined framework relies on IOP-type infrastructures, such as Smart-M3 [19]. It has features for self-management of qualities that have not so far been supported by IOP-implementations. The framework takes care of quality requirements and their changes during system's operation. As stated in [43], support for non-functional requirements is still missing or immature in the existing software engineering practices for developing the smart applications. In [60] representatives of mobile middleware have been compared using context types, context model, and intelligence as evaluation criteria. The conclusion was that the approaches surveyed lack support for heterogeneous environments, they do not mention how to address evolution of ontology, runtime or dynamic context is not considered, and not much contextual knowledge is used to enhance agents. Furthermore, existing approaches have weak support for malfunction recovery and other self-management features.

Two more recent context-aware and self-adaptive solutions for networked devices that are closest to our approach include MUSIC[2] and Hydra[3]. They are both EU projects that have already been concluded. Both MUSIC and Hydra are solutions that are based on Service-Oriented Architecture (SOA). However,

[1] http://www.sofia-project.eu/

[2] http://ist-music.berlios.de/site/index.html

[3] http://www.hydramiddleware.eu/news.php

as self-management middleware only, Hydra uses publish/subscribe-style communication.

MUSIC is also a component-based platform, where adaptation is based on plug-ins. MUSIC especially supports mobile applications. Adaptation is controlled by adaptation managers. One adaptation manager is in charge of one adaptation domain which consists of one master node and slave nodes. The adaptation domain and bindings to the slave nodes change dynamically depending on the movement of the master node or changes in connectivity. Slaves can participate in multiple adaptation domains. Furthermore, non-MUSIC applications can be included for providing and using services as long as they use the same protocols for discovery, binding and communication. Using services outside the adaptation domain requires negotiation of Service Level Agreements (SLA) with the provider.

Developing Hydra applications requires defining functionality of the client and server devices, as well as their self-management support. Adaptation decisions are made by the self-management middleware based on knowledge of clients and servers without negotiation. Hydra-based devices include state machines that allow reporting of their state to other self-management components. Furthermore, message probes are used for the monitoring of live of computing nodes. The message probes also facilitate the monitoring of Quality of Service (QoS). Support for non-Hydra devices in applications is missing.

The MUSIC platform provides a library of sensors that are able to detect violations in the dimensions of the agreement. Furthermore, on the consumer side, the consumer proxy is instrumented with appropriate monitoring mechanisms according to the content of the SLA contract. The service skeleton on the producer side is instrumented with context sensors, which are responsible for monitoring the agreement.

Hydra relies on Web Ontology Language-based (OWL) ontologies in realizing plans. Behavior is represented in Hydra as Semantic Web Rule Language[4] (SWRL) rules, which do not support updates at run time. MUSIC uses ontologies only in design-time and adaptation decisions are based on utility functions. HYDRA defines a Device ontology that includes Service, QoS, and Security ontologies. Ontologies are application-agnostic and provided as part of the middleware. They cover dynamically changing information, such as device states. Thus, although context information (i.e., device states) and QoS are monitored, the domain independent ontologies for context, security, and performance are not provided.

An adaptation framework that uses OWL-based ontology is presented in [3]. Its ontology is domain-specific and for dialoguing between user and a system, such as an automotive dashboard and a digital TV. The framework is dedicated to be used for adapting the interface according to the user-system interaction. Therefore, it is not as widely usable as the presented adaptation framework.

2.2. Situation-based application adaptation

Smart space applications interact with end-users, and therefore context is an essential part of these applications. Handling of the context is presented via the evolution and adaptation management processes which are of equal importance for the architecture of smart environments. The evolution of context is handled by the appropriate context ontology, which includes generic and relevant concepts and is expandable by domain and quality ontologies. The domain means an application, such as managing lighting at home, GuideMe navigation, intelligent wake-up or smart building maintenance. Several studies have been made on context ontologies, e.g., CoOL [55], CONON [58], and SOUPA [5]. The common problem with their usage is that they are either domain specific or include user authentication via policies. According to the comparison [60], CoOL and SOUPA do not support dynamic context. Therefore, a generic approach was selected for defining context ontology for smart environments. First, the levels of context concepts were defined in [2] and used as a starting point. Second, some parts from the upper context conceptualization [52] were exploited. Third, four dimensions were defined: physical, digital, situation, and user context, which were validated by applying them to a simple demonstrator implementation [37]. Fourth, some concepts of the SOUPA ontology [5] were mapped to the appropriate context levels. Finally, the concepts of each context level were enhanced based on experiences in smart environment application development. The recent version of Context Ontology for Smart Spaces, CO4SS [38], has six dimensions: physical, digital, situational, user, social, and historical context.

The use of ontologies follows the style of the semantic web; ontologies describe context information and its associations and provide means for reasoning and inference. RDF and OWL [36] are used for describing context information and context ontologies,

[4] http://www.w3.org/Submission/SWRL/

respectively. These descriptions are combined with middleware or a framework for providing a complete solution for context management.

The context-awareness micro-architecture [37] is the solution for managing adaptation based on the context in the smart environments. It consists of three types of software agents: the context monitoring, context reasoning, and context-based adaptation agents. These agents share information via the semantic database, i.e., the SIB described above. Micro-architecture is a set of patterns used together to realize parts of a system or subsystem [1]. Micro-architecture consists of software agents and is seen as a building block for piecing together well-known, cohesive portions of overall architecture.

The context-awareness micro-architecture has been used i) to activate required functionality according to the rules and existing situation(s) [37] and ii) to map context and domain-specific ontologies [39]. The rule-based reasoning is based on a set of rules and a set of activation conditions for these rules. In practice, the rules are elaborated if-then-else statements that drive activation of behaviors, i.e., activation patterns. One of the rare solutions for rule-based context reasoning is presented in [7]. Therefore, the existing technologies were explored for implementing behavior into software agents running on smart spaces. These approaches allow treating behavior descriptions as semantic data in their own right – as data that can be stored into a database (e.g., an SIB), queried, exchanged between agents, and so on. The main purpose of using such declarative behavior descriptions is to enable a greater level of dynamism in smart environments with respect to devices, and for the agents running on those devices to be able to modify their behavior according to the context situation at hand. This allows the agents on mobile devices to, for example, change the security mechanism used or adjust processing power used when the battery is low and no critical applications are running. Another example is to modify the behavior of the application according to the place where the user is using the device. Especially in the latter case, it is reasonable for a software agent not to be pre-programmed with all behaviors for all possible locations but rather access relevant rules from the local smart space.

2.3. Rules description languages

In smart spaces, applications can be written by many languages. Thus, the selected behavior description language shall be a generic solution that can be applied with all application programming languages. In the end, the use case and the needed functionality determine what rule language to use; some are good for simple cases and others for complex cases. Table 1 presents a comparison of three approaches considered promising for smart spaces; namely, Smodels [27,49], Recipes [23], and SPARQL [53]. SWRL is left out of the comparison because it is not a W3C standard as SPARQL is. Basically, all of the options use rules that can be stored into the semantic database, and the actual behavior modification triggers (actions to be executed) can also be inserted into the database. However, only Recipes are an RDF–triples-based language. Smodels rules can only be stored in the database as literal objects of triples. Given that there might be a size limit on a triple's object size, the size or complexity of the rule that can be stored into a semantic database implementation may be limited. Another option, of course, is to store as triples only the URLs of external documents where the rules are located. SPARQL, although a W3C standard language for querying RDF, does not fully follow the RDF-triple data model (its CONSTRUCT clause is triples, its WHERE clause is mostly triples, with the exception of special constructs like FILTER or UNION, and the way as clauses are combined in a query is not triples). Therefore, SPARQL queries as such can only be recorded as literals. However, SPARQL RDF-serializations exist and are used in practice. For example, the most widely known is the SPIN SPARQL Syntax[5], which is a central element of the SPIN framework.

Table 1
Comparison of rule languages

Rule language	Smodels	Recipes	SPARQL
Reasoning execution	Client	Client	Client/Server
Is a standard			x
Supports action-part	x	x	x/o
Storing rules as	string literal or RDF triples	RDF triples	string literal or RDF triples
Complexity of rule	complex	moderate	easy or moderate

An approach to improve the intelligence of pervasive middleware is presented in [60]. The authors share the same vision of "intelligence" in which the application needs to be supported to recognize changes affecting it and adapt behavior accordingly.

[5] http://www.spinrdf.org/sp.html

The approach is based on OSGi [35] and uses the Self-Management Pervasive Service (SeMaPS) ontologies, which consist of a set of context ontologies.

The approach can use dynamic context information and static rules specified with SWRL. Dynamic rules are not supported and user, historical and social concepts are not included. SeMaPS seems to concentrate more on devices and their states and the networking between them.

Our experience from demonstrator implementations convinced us that rules processed in smart spaces are rather simple although their processing is frequently required. SPARQL allows the rules processing to be done on either the client side or the server side. This reduces the amount of information that has to be transferred between the client and the SIB, since it is not necessary to transfer all the information in the rules to the client side before executing the rule. Therefore, SPARQL was selected to be used as the rule description language. This was done mostly because it is the only standard-based approach.

A standard-based approach is assumed to give better support for ontology evolution management. However, a few open issues still exist: i) how to map the context ontology with quality ontologies, and ii) how to use the context-awareness micro-architecture to manage qualities at run time.

2.4. Run-time quality adaptation

Self-adaptive software is a closed-loop system with a feedback loop aiming to adjust itself to changes during its operation [45]. In MAPE-K style [22] the control loop of self-adaptation is decomposed into monitoring, analyzing, planning and execution. Monitoring means collecting the data needed for adaptation from the system under interest or its environment. Analysis is the phase where the collected data is combined to form proper metrics, and also predict future states. Planning is the control phase, where the necessary action is decided upon. Planning can be an aggregate of local and global level reasoning; the local requirements have to be filled by individual capabilities and the global requirements by the service composition [29]. The planning phase may also include negotiation.

Execution of the self-adaptive actions can occur on two different levels [30]: i) the resource management level performs application-neutral adaptation, and ii) the service management level is responsible for the application adaptation. Adaptation actions include changing values of parameters, changing the structure of components (i.e., compositional adaptation), and re-organization of the topology of the application across multiple platforms and changing the workflow or the functionalities provided, which leads to behavioral adaptation [34,45].

The phases of the control loop use a common knowledge of the system and its environment. The knowledge can be represented using [28]: i) modeling languages, ii) interface languages, iii) application interfaces and deployment descriptors, and iv) mathematical models. Moreover, semantic service descriptions, including ontologies of service context, functionality and quality capabilities, are especially useful in open pervasive environments, as it is unreasonable to assume that service developers will use identical terms when describing services [29].

In this work, the knowledge is related to the quality of the information and ontologies are used for defining it. Performance ontologies have been developed, for example, for grid workflows [57] and semantic web applications [25]. However, the individual requirements of each application domain affect the composition of the ontology so they were not directly reusable in our case. QoSOnt [8] is a QoS ontology for service-centric systems. QoSOnt combines quality and services whereas the quality is combined with information in the framework defined in this paper.

Reliable run-time quality management requires that there be a common understanding of what is measured and how it is measured. In order to achieve that, the software measurement ontology [17] has been used both in the information security ontology (ISMO) [14] and the runtime performance management ontology (RPM) [44] as a basis.

The micro-architecture created for security adaptation [13] contains separated parts for monitoring changes in security requirements and for monitoring the fulfillment of the required securities. In that micro-architecture, the usage of ontology was tightly coupled inside the model. The enhanced micro-architecture was developed, which follows the MAPE-K style and where ontologies are separated from the architecture to the other interoperability level [15]. In this work, micro-architectures for performance and security adaptation and related ontologies have been merged into a complete adaptation framework.

Dynamic environment requires for flexible approach for making adaptation decisions. The use of preference information allows for the behavior of applications at choice points to be manipulated [18]. We adopted similar approach for managing individual quality adaptation objectives. In case of conflicting

needs, the final adaptation reasoning chooses the configuration option that best fulfills all the quality objectives.

3. From requirements to smart environment architecture

3.1. Knowledge creation

When the interoperability platform (IOP) for the smart environment started to develop, there was a lack of understanding of the interoperability models, appropriate architectural styles, and existing solutions. Therefore, the following steps were carried out for achieving the required knowledge on the software technologies related to situation-based and self-adaptive applications:

First, the existing interoperability models were studied and the conclusion of six levels was reached as described in Section 3.2.

Second, starting from the 56 application scenarios defined for three types of smart environments, personal spaces, smart indoor environments and smart cities, requirements were transformed from them and classified into two categories: quality and functional requirements (see Section 3.3).

Third, those requirements were carefully analyzed and prioritized as the scenarios in the second step according to the criteria: i) the maximum business impact, and ii) the fast and low-risk realization criteria. As a conclusion, the IOP principles were defined based on the high-priority IOP requirements (16 quality requirements and 12 functional/non-functional requirements), and used as a style while architecting the IOP [32]. These two sets of requirements were also used for defining the evaluation criteria for IOP instances. The evaluation results of the different IOP implementations have been reported in [51]. Due to limited space, only the IOP principles (Table 2) are introduced in this paper.

3.2. Interoperability of smart environments

Interoperability models proposed or adopted in existing platforms vary depending on which interoperability levels are considered, how interoperability is conceived and the technical solution adopted. The Connection, Communication, Consolidation, Collaboration Interoperability Framework (C4IF) [41] exploits the concepts of language theories, such as the language form, syntax, meaning and use of symbols and interpretation. C4IF maps the linguistic concepts to the interoperability levels as follows:

- Connection interoperability is the ability to exchange signals and the channel used as an object of integration without knowing anything about content. Connection interoperability is a prerequisite for any interaction between physical entities.
- Communication interoperability is the ability to exchange data and use information as an object of integration, i.e., format and syntax of data but without knowing the context in which data is used. Communication interoperability is provided by low level interaction capabilities of physical (hardware or software) entities.
- Consolidation interoperability is the ability to understand data and its meaning and use information as an object of integration but without knowing how it is used. Consolidation is provided between software entities, e.g., components and services by means of semantics, such as metadata and service ontologies.
- Collaboration interoperability addresses an ability to act together and uses processes/tasks as an object of integration. Collaboration interoperability is achieved between tasks, processes, etc. if these software entities are able to adapt their behavior by taking into account not only the context, but also other actors' behaviors related to the same collaborative activity.

The M3 (multi-vendor, multi-device, multi-domain) is baseline architecture for the Smart-M3 architecture [19]. The M3 concept distinguishes three interoperability levels: device, service and information.

The interoperability levels were further elaborated in order to match them better to the development of smart environments and their applications. Figure 1 presents the results of this exploration. Basically, the three first levels (from bottom to top) are quite similar to the levels of the C4IF and M3 models. However, they have been named according their objectives as follows: connection interoperability, communication interoperability and semantic interoperability.

The rest of this paper will focus on the two upper levels: dynamic and behavioral interoperability and the means for achieving them. Conceptual interoperability is also supported by reference architecture, the patterns and ontologies used for realizing the adaptation management architecture. However, our earlier works that concern styles, patterns, and the knowledge-based software engineering methodology [31,33], are also referenced.

Table 2
The IOP principles

IOP principle	Definition
Shared information	The IOP manages a shared information search domain called Smart Space (SS), which is accessible and understood by all the authorized applications. This information is about the things existing in the environment or about the environment itself. The information is represented in a uniform and use case-independent way. Information interoperability and semantics are based on common ontologies that model information.
Simplicity	The IOP deals with information. The IOP information level is use case agnostic.
Service	An SS is a service, offered by a service platform and intended for the sharing of interoperable SS information. Each application may interface to one or more SSs through a Smart Space Application Protocol (SSAP). Use case-specific functions may be performed at the service level before joining the SS.
Agnostics	The IOP is agnostic with respect to the adopted ontology, application programming language, service platform exposing the SS, communication layer and hosting device/system.
Extensibility	The IOP provides functionalities to insert and remove the information. IOP functionality may be extended with domain ontologies and with information manipulation services. If these services become commonly usable, they are called "IOP extensions."
Evolvability	The IOP should support the addition of new applications. This principle envisages that the IOP provides the means to implement software that adapts to changes in SS without changing code.
Context	Context management is an IOP extension, according to the extensibility principle. The IOP should enable the aggregation of interoperable information for the benefit of application usability and IOP performance. As the information returned by the IOP depends on the query and available information, the ontology is to define context semantics. Context may be managed and used both at the information level and service level.
Notification	Applications may subscribe to be alerted upon a context change.
Usability	User interaction management may become an IOP extension, according to the extensibility principle. The ontology defines the semantics of interaction events. The interaction between the users, their environment and Smart Space Application (SSA) may be managed both at the information and service level.
Security & trust	Security, privacy and trust management is an IOP extension, handled both at the service level and information level. Appropriate ontologies define whether the IOP is required to respect privacy, enforce authentication, and access control policies at finer granularity than SS itself, or if the shared information integrity, confidentiality, and trust need to be provided.
Legacy	Legacy devices and systems access and exchange information with the SS through a simple use case independent protocol (SSAP). Such exchanged information is modeled by Domain Ontologies. Legacy devices may provide information to the SS and subscribe to information from it.
Scalability	The IOP should scale with respect to the number of users, devices, and resources available on each device, as well as the amount of information stored in the SS, and the number of SSs.
Performance	Performance monitoring is an IOP extension. Performance of IOP realizations and SSAs should be evaluated at the development time and be measurable at run time. The criteria for run-time performance monitoring should be defined through the performance metrics ontology.
Reliability & availability	The reliability and availability of every IOP instance and of SSAs should be evaluated at development time and be measurable at run time.

3.3. From requirements to principles

As previously mentioned, the requirements were classified into quality requirements and functional requirements. The quality requirements were related to execution qualities: information security, availability, performance, reliability, adaptability, and usability; or evolution qualities like integrability and extensibility. The functional requirements concerned communication styles, and abilities to evolve, be dynamic, be proactive, be context-aware and be applicable within heterogeneous environments.

As a conclusion of discussions carried out in several workshops, the project partners involved in development of the IOP agreed on its main objective: to provide an infrastructure that assists users with added-value interoperable information about things existing in the user's environment. Therefore, the IOP reference model is to be defined at a high abstraction level and it shall be simple and agnostic with respect to i) the use-cases, ii) information and iii) the physical environment. These were expected to enable the level of extensibility required to support multi-domain and cross-domain applications. The expected applications shall support situation- awareness, spontaneously start when required, and have the potential for significant market penetration and socio-economic impact. As a result, fourteen IOP principles were distilled from the requirements (Table 2). As can be seen, the quality and adaptability principles

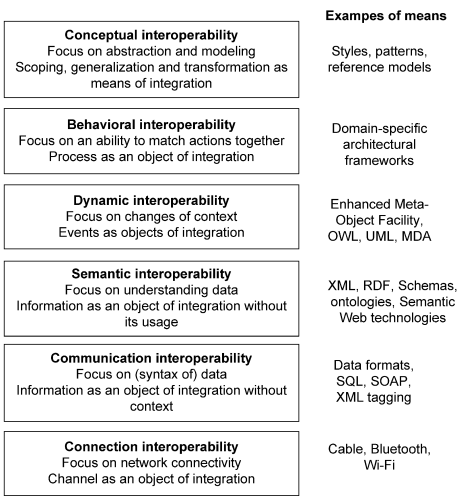

Fig. 1. Interoperability levels of smart environments.

were agreed to be produced as IOP extensions. These extensions are the main contribution of this paper. However, all these principles summarize the significant requirements and specify the architectural style the IOP has to follow.

3.4. Smart environment architectures

In [34,56], the authors survey the existing architectural styles, frameworks, and approaches and their ability to manage evolution and adaptation. The framework used for the comparative analysis has two main parts:

1. *Evolution management* that takes care of i) architectural descriptions, i.e., models, ii) evaluates the consistency of models and system integrity, iii) provides methods for realizing models, and iv) accepts changes and collects observations; and
2. *Adaptation management* that focuses on runtime properties of the instantiated architecture. The adaptation management architecture i) monitors observations and evaluates measurements, ii) changes the plan based on observations, iii) allocates changes to architectural assets, and iv) accepts changes and collects observations.

Thus, these two management processes are cyclic and share the last activity – accepting changes and collecting observations.

However, their cycles are different: adaptation is made at run time without user intervention but evolution is managed through models that are kept as static as possible. In the architectures of smart environments, both adaptation and evolution are equally important, as evolution management is for providing models that make smart space assets interoperable, and the adaptation management has to provide the means to fulfill functional and quality requirements in the best possible way whatever the situation, i.e., tolerating changes in user requirements, changes in physical environments, and changes in the internal capabilities of the smart environment's infrastructure. Thus, interoperability depends on how well the evolution management is handled, and the situation-awareness of applications depends on how well the situations are identified and how proactively adaptation actions could be made. In the best case, the user does not recognize the adaptation at all.

In IOP-type infrastructures information interoperability is supported by the means of Semantic Information Brokers (SIBs). SIB is a Resource Description Framework (RDF) [59] database to which software agents (called knowledge processors, KPs) can connect and exchange information through an XML-based (eXtensible Markup Language) interaction protocol called Smart Space Access Protocol (SSAP). KPs communicate with the SIB through a Knowledge Processor Interface (KPI). KPI provides the available SSAP operations (join, leave, insert, remove, update, query, subscribe, and unsubscribe) to be used when communicating with different SIBs.

There are multiple different instantiations of the SIBs for different purposes: Smart-M3 for resource rich devices and systems without real-time requirements [48]; SOFIA Application Development Kit (ADK) for simulation purposes with Java; and RDF Information Base Solution (RIBS) [54] for resource constraint devices with strict security and performance requirements. KPs consume and produce RDF triples from and into the SIB according to the ontology used.

In this paper, communication protocols are assumed to be able to manage connectivity and communication interoperability, and that any of the realizations of the IOP concept, named Smart-M3, SOFIA ADK or RIBS, manage the information interoperability that concerns information exchange (KPI, SSAP and SIB) and interpretation of information semantics according to the predefined RDF Schema. The schema used is described as ontology by the web ontology language (OWL) and stored into the SIB. The contribution of this paper is what has been developed

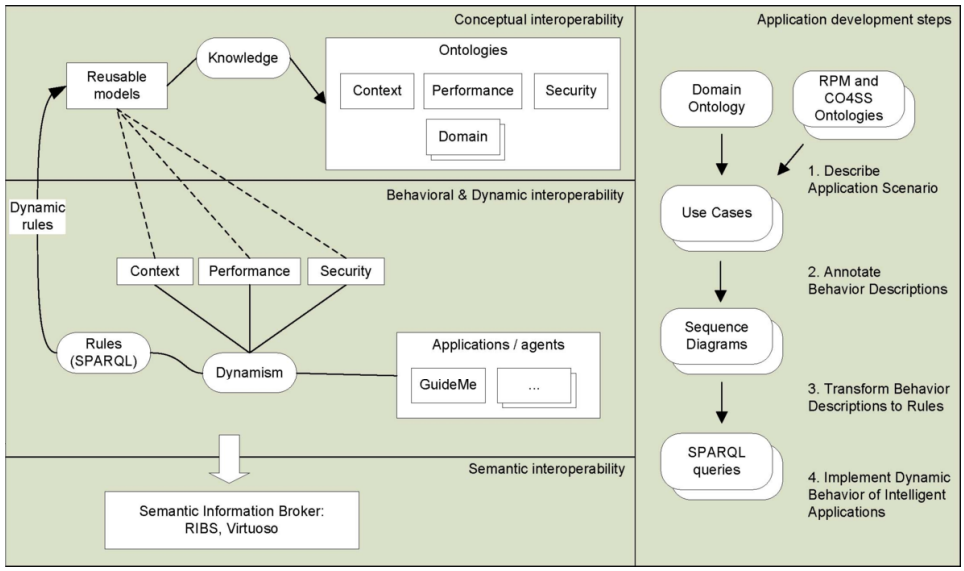

Fig. 2. Overview of the adaptation framework and the application development process.

on top of these existing interoperability solutions: the evolution management solutions for context and behavior interoperability levels and the adaptation management solutions that are totally missing from the existing IOP solutions.

4. Adaptation framework for situation-based and self-adaptive applications

4.1. Overview of the adaptation framework

An overview of the adaptation framework (left-hand side) and the application development process (right-hand side) is presented in Fig. 2. The framework is mapped to the interoperability levels of smart environments shown in Fig. 1.

The framework consists of three levels, which provide conceptual interoperability, behavioral and dynamic interoperability, and semantic interoperability. Conceptual interoperability is achieved by the knowledge, represented as ontologies. Context, performance and security (instantiations of the MAPE-K pattern) provide information to applications for handling their behavior. The rules are the basis for the dynamism to enable behavioral interoperability with the platform used.

The application development process, based on the adaptation framework, is divided into four steps, as shown in Fig. 2. The process guides starting the situation-based application development by describing the scenario with related behavior and the information. The ontologies are used for identifying the relevant information. The ontologies used are domain, Run-time Performance Management (RPM), Information Security Measuring Ontology (ISMO), and Context-Ontology for Smart Spaces (CO4SS).

First, the application scenario with the related behavior is described by textual and graphical notations. A standard way of describing an application scenario by text and use cases is preferred. As a result, a set of software agents are identified and collaboration between agents and users is described by a (set of) use case(s). Second, the behavior is further described by Message Sequence Chart (MSC) diagrams with rule and ontology annotations. Third, the behavior description is transformed into SPARQL queries. This is made by the developer by exploiting the MSC diagrams and the defined ontologies to create SPARQL queries. Fourth, the developer handles the dynamicity of the space by ensuring that the adapted intelligent objects (see Fig. 8) publish their functionality and allow for controlling them. In this step, the developer also decides how many adaptation agents are used

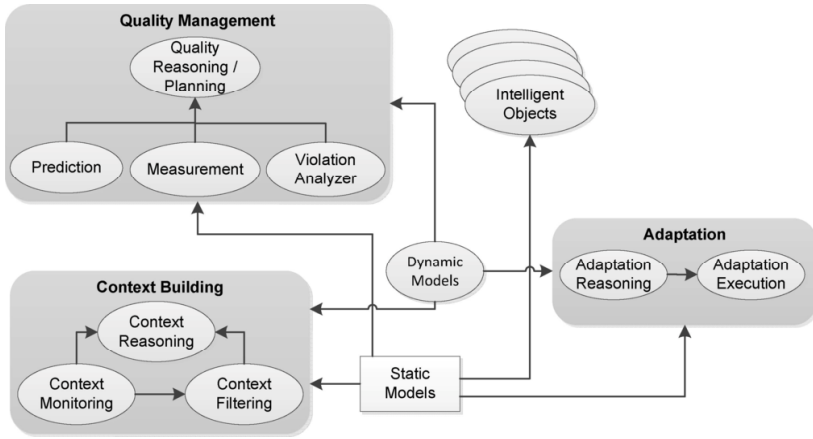

Fig. 3. The adaptation framework for situation-based and self-adaptive applications.

from the adaptation framework and how they are deployed on the client and server sides.

The adaptation framework supports the evolution of smart environments and provides reactive and/or proactive behavior when the application context changes. Evolution management is dealt with by a set of static models, i.e., ontologies that define context, quality (as performance and security) and domain-related concepts and properties in a standardized way. Thus, the environment is able to tolerate changes if there is a common vocabulary that defines the semantics of concepts, their properties and relationships. With the common knowledge, i.e., standard definitions of concepts and their meanings, it is possible to achieve stability in the dynamic systems. Moreover, the clear scoping of ontologies makes it possible to evolve these ontologies separately and update the systems according to the newest set of ontologies.

There are four kinds of ontologies that are required: context, performance, security, and domain ontologies. The context ontology is for context-building. The performance ontology is for defining the objective, measures, and means for performance monitoring and analyzing the measurement results accurately and unambiguously. The security ontology has similar targets as performance ontology but for information security. The domain ontology is for adapting the generic concepts of context, performance, and security to the concepts of the application domain. Thus, the domain ontology is only changed when the adaptation framework is applied to a new application domain.

The adaptation framework exploits the architectural patterns – e.g., a monitor, adapter, interpreter and reasoner – for providing a multi-domain architectural solution, the reference model for run-time quality management. Thus, the architecture of the adaptation framework exploits reusable models on two levels: on the ontological level and in the reference architecture that is adapted to different smart environments by changing the domain-specific concepts to other ones. Therefore, the mapping of the context, security and performance ontologies to the domain ontology needs to be made in a generic way that could be applied in any context where situation-based and self-adaptive applications are needed.

Another set of ontologies, i.e., rule ontologies, is required for changing behavior of the smart space and its applications at run time. Behavioral changes are made by analysis models and adaptation algorithms that are defined as a set of rules or rule ontology. The application framework introduced in this paper is a logical architecture (see Fig. 3) and its realization in a smart space may embody a set of context-building elements as well as the adaptation elements.

In the following sections, the elements of the adaptation framework are introduced in more detail. First, the static models, i.e., ontologies, are introduced. Second, the elements of the adaptation framework that exploit the defined knowledge representations are introduced: context building, quality management and adaptation. Third, the dynamic models used for achieving situation-based behavior for applications are explained.

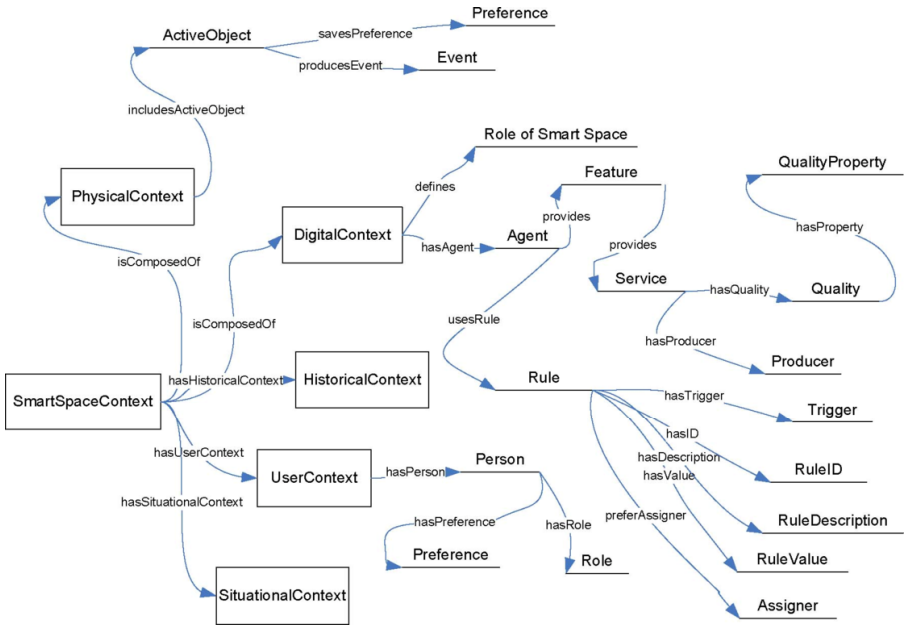

Fig. 4. Adjusted CO4SS ontology.

4.2. Static models

4.2.1. Context ontology

Context ontology for smart spaces, CO4SS, defines the generic concepts related to smart spaces. The context ontology is static in a sense that it is not updated at run time but only at design-time. Thus, the objective of the context ontology is to support the evolution management of the smart environment: all smart spaces and their applications 'understand' the common language defined by the CO4SS. Thus, the context ontology is used as a foundational ontology, which application specific concepts are mapped. In the adaptation framework, the CO4SS concepts are also used together with the quality concepts. The CO4SS needs two new properties, a Trigger and an Assigner, for its Rule class due to the adaptation framework and mapping with the performance concepts, as introduced in Fig. 4. These new properties are general by their nature. Relevant context information for information security [13] is presented in Fig. 5. Only the most relevant context concepts for run-time security and performance management are presented in Fig. 4. Despite the tuning, the CO4SS is still general ontology to be used as a base ontology when creating the semantics for the smart spaces.

Physical Context describes the information coming from the physical and real things, such as sensors or a mobile device. Preferences and the role (e.g., visitor, worker, customer or owner) of the users are saved to their smart spaces via devices. Physical Context also represents real events, e.g., output coming from the sensors or devices (as pictures, text messages, videos, e-mails), location updates, failures or connections made by objects that have arrived (or been added) to the smart environment. These events relate to the Situational Context. The adaptation in the smart space application is mostly triggered by the events.

Digital Context consists of agents, features, and rules, and the feature is respectively formed from services and qualities. The rules define the situations, i.e., the behavior of the (smart space) application. User Context defines a person with roles, preferences, and friends. Different kinds of activities can also be represented, e.g., morning activities (showering, breakfast, morning newspapers, etc.), travel activities (work-related travel, holiday travel, etc.), and exercise activities (jogging, the gym, different games, etc.). These activities can be stored and used later on to make reasoning according to different situations, e.g., how long it usually takes to perform morning activities.

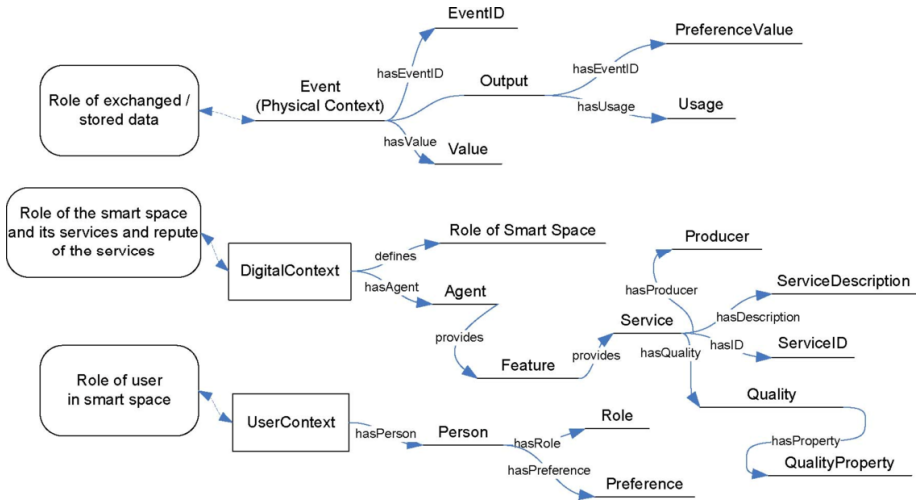

Fig. 5. Context information for security.

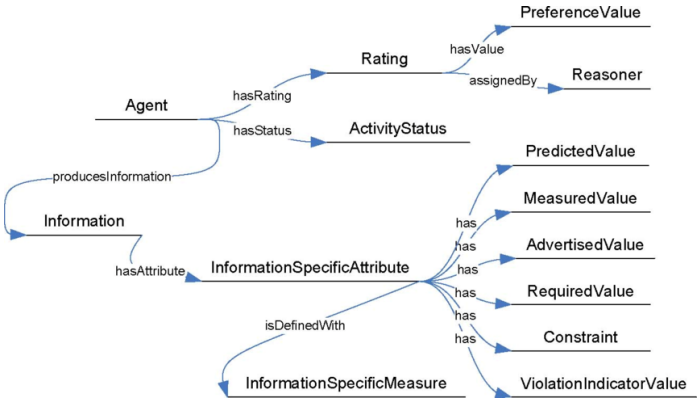

Fig. 6. The refined RPM ontology.

Situational Context is i) influenced by events coming from physical things, ii) affected by the preferences of the users and social groups, and iii) defined by the rules. Thus, the rules guide the recognition of situations.

4.2.2. Quality ontologies

The RPM ontology defines the concepts that are required for managing performance at run time. It combines the smart space concepts such as agent and information to performance concepts such as quality attribute and measure. In IOP infrastructure, applications are composed dynamically of independent agents. Thus, the quality objectives of the applications also depend on the situation. Consequently, the adaptation framework requires addition of a new class – Rating (see Fig. 6). In case of multiple options for an information producer service, performance reasoning identifies the relative preference of each option and by that allows separation of concerns for management of individual qualities.

Information Security Measuring Ontology, ISMO, measures information security and provides a solid way to present security measures for software designers and adaptable applications [14]. The ISMO combines existing measuring and security ontologies

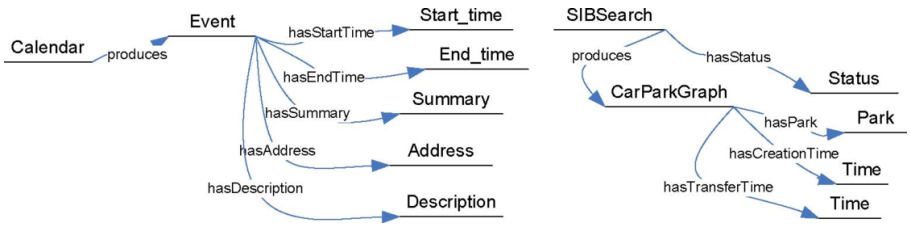

Fig. 7. An example of domain ontology.

and instantiates it through example measures. Figure 5 introduces the most relevant context needed from CO4SS for run-time security management. The rounded rectangles present the information needed for security adaptation. The dashed lines connect the abstract information to the relevant CO4SS concepts.

4.2.3. Domain ontology

Domain ontology, such as that illustrated in Fig. 7, describes the application-specific concepts, properties and relationships. The domain ontology concepts are to be mapped with the CO4SS, ISMO, and RPM ontologies. The mapping is done by comparing the domain concepts between the context, security and performance concepts to find commonalities or same concepts. The use case dictates what kind of mapping has to be done between these ontologies.

A situation-based agent was developed for supervising the maintenance scenario of a building [39]. The maintenance scenario starts when, for example, a sensor detects water on the floor. The smart maintenance system with the ontology is legacy software that was enhanced with new features, consisting of the Context Selector and Context Monitor. In order to perform context-aware supervision, the context and maintenance ontologies are aligned to get the new feature to be interoperable with the legacy (smart maintenance) system. After alignment, a shared context, i.e., a graph, is in place to supervise the progress of the maintenance scenario. The smart maintenance ontology is used herein for detecting faults, scheduling interventions and supervising tasks of maintenance people.

A new way for bounding the context information relevant for the context monitoring agent in question was introduced [39]. The context is selected with a GUI (Graphical User Interface), called Context Selector [16]. The Context Selector uses the context ontology (CO4SS) by reading its concepts from the file that is in the OWL [36] format. Thereafter, the Context Selector maps the main context concepts to the corresponding smart maintenance concepts. The Context Selector saves the selected context to the semantic database, such as RIBS [54], from which the Context Monitor is able to reach the context it needs to follow.

4.2.4. Ontology mapping

The concept-level mapping between the CO4SS and RPM ontologies revealed that they have only one concept with a similar name. That was the *Agent* class, but with different properties. In this kind of case the mapping is seen to require performance using an OWL property (i.e., owl:equivalentClass) to introduce these Agent-classes as equivalent but not necessarily the same concepts. Another OWL property, owl:sameAs, is for linking an individual to an individual. The relevant context concepts have been previously recognized for security monitoring purposes [13]. Therefore, the concept level mapping between CO4SS and ISMO were no longer necessary to perform.

Domain ontologies are seen as 'configuration models' for the CO4SS, ISMO, and RPM ontologies that are generic and applicable for any application domain. For example, agents and information in the RPM ontology are super classes to the domain-specific agents and information. In practice, the mapping will be handled by the dynamic models, i.e., configuration parameters, analysis models, and adaptation rules, and the interpreters developed for this purpose.

4.3. Knowledge execution elements

Next, the elements of the adaptation framework are introduced. The execution elements are introduced with the GuideMe navigation service in Fig. 8. GuideMe consists of intelligent objects that are legacy adapters or traditional agents that do not use the adaptation framework: SIBSearch, WebSearch, Navigator, and CarParkService. The *Context building*

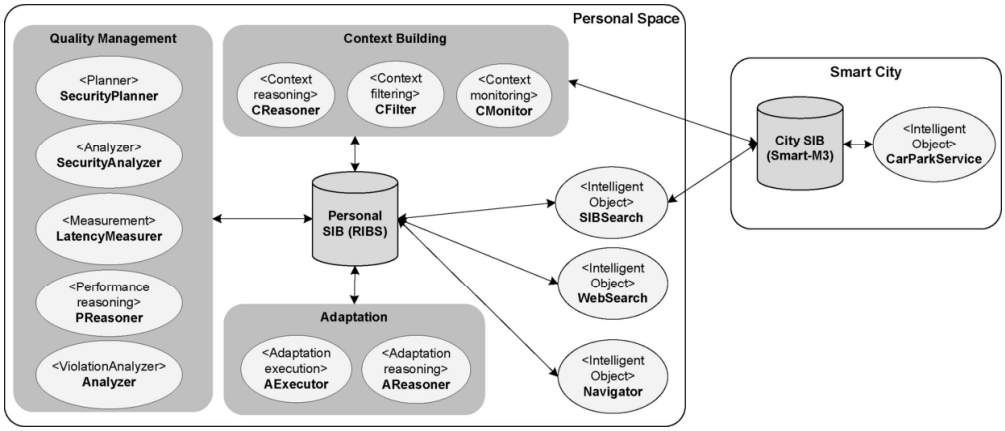

Fig. 8. Use of execution elements of the Adaptation Framework in GuideMe.

element includes agents for context monitoring (CMonitor), context filtering (CFilter), and context reasoning (CReasoner).

The *Quality management element* has agents for measuring, predicting, reasoning or planning, and analyzing. GuideMe has LatencyMeasurer, PerformanceReasoner (PReasoner), SecurityPlanner, and Analyzer agents. Thus, own agents are instantiated for each quality. The *Adaptation element* has agents for adaptation reasoning (AReasoner) and adaptation execution (AExecutor). The use of these elements in the GuideMe application is presented in Section 5.

4.3.1. Context monitoring

Context monitoring is configured to collect certain information that is relevant from the viewpoint of the used context, e.g., for security monitoring or energy consumption. It shares the monitored content via the semantic database, such as SIB. The configuration is herein given by the ontology as a set of RDF triples or by a SPARQL query. The configuration stored as an RDF-graph (or string literal in the case of SPARQL query) in the SIB scopes the context to be monitored. The configuration can be given in design time, at run time or when the context monitoring agent is instantiated, i.e., in the startup phase.

4.3.2. Context reasoning

Context reasoning is used for interpreting the rules, which can be set during the design phase or they can be given at run time or when the context reasoning is activated. Thus, the context reasoning is configurable. The output of context reasoning is used either for defining context for quality management purposes or for adaptation.

The context reasoning exploits the information monitored by the context monitoring. In addition, the context reasoning can use other information from the semantic database. Usage of the information depends on the rules. The other information can be current or historical data. The current information can be, for example, user preferences related to the performance of the application. Naturally, the historical data is collected to the SIB beforehand. For example, in a smart home, historical data may include all typical activities and habits of the family members on working days and free days. This kind of historical data is essential to be able to proactively process the context of the future, and not only the context of the present moment. The context reasoning can exploit any external historical data that is relevant and available to it. Thus, reusability of existing information is taken into account in the context reasoning.

4.3.3. Context filtering

Context filtering is intended to be used for tuning the inferred context according to the filtering parameters, e.g., the user's privacy filter or the membership of a social community allows a wider visibility to certain information. The context filtering and context reasoning work alike: both are configurable. Thus, the filtered context is a specialized view of the inferred context and can be used in some instantiations

of the application framework. The quality management uses the filtered context when it is necessary.

4.3.4. Quality measurement

Quality measurement provides the measured quality that defines the current status of the quality of intelligent objects. What measurements are produced depends on the requirements, user preferences, and availability of means for measurement. Performance measurements are related to the time behavior and resource utilization of intelligent objects. Security measures observe security-related attributes in the environment and in the intelligent objects. These attributes describe the used security mechanisms and protocols, and in addition, threats from the environment. Measures can be used as base measures or they can be further processed to derived measures, such as statistical measures. Measuring results are separated from reasoning so that one measure can be utilized in different types of reasoning.

4.3.5. Quality prediction

Quality prediction uses the measurement history in order to forecast the future status of the system and its environment. The predicted quality can be used instead of measured quality to ensure continuously acceptable behavior.

4.3.6. Quality reasoning

Quality reasoning rates the adaptation options and gives the information regarding quality preferences to the adaptation reasoning. Reasoning is performed the first time quality (performance and security, especially in this paper) management is started and after that when the context of quality management changes or the violation analyzer informs that the particular quality is no longer at an acceptable level.

Reasoning is based on the analysis models. The required quality denotes the requirements of the intelligent object that is using the services or information of other objects. The advertised quality is informed by the intelligent object that is producing information or services. Advertised quality can be based on measured values, but it can also be fixed already at development time.

4.3.7. Violation analyzer

The violation analyzer notifies of violations of the requirements. The notifications trigger quality reasoning. The violation analyzer monitors the requirements and compares them to the advertised, measured, or predicted qualities, depending on the situation. Changes in any of those may result in that quality (performance and/or security) is no longer at an acceptable level.

4.3.8. Adaptation reasoning

Adaptation reasoning makes the decision regarding which adaptation option to select based on the situation context, quality options, and the related adaptation rules. Adaptation reasoning unifies the reasoning activities from the context-awareness micro-architecture and run-time performance and security management. The output of the adaptation reasoning is the decision for the adaptation.

4.3.9. Adaptation execution

Adaptation execution follows the decision made by the adaptation reasoning and supervises the actual adaptation actions that are needed. The adaptation execution may require knowledge of the platform and the environment in order to be able to perform the required adaptation in a robust manner. An adapted object, i.e., the part of software adaptation that is the target of adaptation, is the result of this activity.

4.4. Dynamic models

The behavior of the situation-based and self-adaptive applications cannot be fixed at design-time. Hence, there is a need for the dynamic models to manage adaptations. The dynamic models are a set of rules or parameters to configure the behavior of the applications in the instantiation phase and/or at run time. A set of rules can be assigned to a specific agent (execution element) and when the rules are changed, the agents automatically start to use the new rules. Rules processing is done on the semantic database side as the agents make SPARQL queries or are notified based on their needs. The usage of notifications assumes that the SIB supports a publish-subscribe mechanism. In the subscribe case, the result of a rule triggered is returned to the agent in hand. The agent can then decide what to do with the result or simply mediate the result to the target database, if needed.

The rule-based dynamic models are further distinguished to analysis models and adaptation rules. The analysis models are used to produce information that can be used to check if the quality requirements are fulfilled. The adaptation rules are used to define what to do in order to fulfill the set quality requirements. Next, the three dynamic models are presented: the analysis models, the adaptation rules, and configuration parameters.

4.4.1. Analysis models

Analysis models are reusable assets that are defined as part of software measurement ontology [17] and applied to different quality attributes such as security and performance. An analysis model is used, for example, when an indicator, such as a password type, is calculated to be used in the password strength. The analysis model is a statement described as a set of rules. These kinds of statements are very close to the natural English language, and because of that they are easy to use and update without extensive knowledge of programming. The use of a natural language helps in concentrating on the content of the models, i.e., the use of domain knowledge without knowing how to model knowledge. The analysis models are used for analyzing the status of the quality properties and processing temporary results for decision-making. The analysis model can be thought of as if-then-else statements. In the adaptation framework, the analysis models are instantiated as standard SPARQL queries.

4.4.2. Adaptation rules

The functionality of the situation-based and self-adaptive application is adjusted according to the adaptation rules, the context, and the measured or predicted qualities. The adaptation rules may include a set of options, limitations, and intelligence for tradeoff decision-making. The adaptation rules are elaborated previously as if-then-else statements in i) the situation-based and self-adaptive application development, and ii) in the run-time performance and security management. Within the adaptation framework the rules are implemented as standard SPARQL queries. These rules are used by context reasoning, quality reasoning, and adaptation reasoning.

4.4.3. Configuration parameters

The configuration parameters are used to configure the monitoring activities of the context monitoring, and are defined by the application designer. The configuration parameters can be updated at run time because the parameters follow the ontologies used. The configuration parameters can be given by the ontology, i.e., a set of triples to match, or by a SPARQL query if the monitored data is more complicated. The idea is that the context monitoring recognizes the current status of the context information and reports this to the semantic database. Later on, the reported information can be used in decision-making and filtering activities.

Fig. 9. Setup of GuideMe.

5. Situation-based and self-adaptive application development

The solution, proposed in this paper, is used to create the GuideMe application that demonstrates usage of the adaptation framework and how the framework helps in the application development. GuideMe uses an external service that produces car park information[6] available in the city that the car is approaching (Fig. 9) and enhances the overall navigator operation with this information and the context information from both the user's personal space and the city space.

5.1. The case study – GuideMe

The GuideMe uses Smart City services to enhance normal navigational service used in the car. When the user is located in the car the GuideMe automatically fetches the meeting place from the user's personal space and selects the closest parking space available. The car park information, the address of the car park and availability of free parking space at the car park, is available either on a web page or in a city smart space (a city SIB). The navigator receives that information from the personal smart space (a personal SIB).

The closest parking space is reasoned according to the context information received from the car park information service (i.e., vacancy, car park location, freshness of the information). When the service pro-

[6] Car park information in the city of Oulu: http://www.oulunliikenne.fi/autoilu/pysakointi

vider's quality decreases, the rating of the services is updated by the quality management and the service used is changed by the adaptation reasoning.

The personal SIB, as context storage, is located in the user's mobile phone (Nokia N9), which also functions as a navigational device and includes the necessary execution elements for context building, quality management, and adaptation. The city SIB uses RIBS [54] as the semantic information broker and runs on an Ubuntu Linux computer. See the illustration in Fig. 9.

The GuideMe specific agents, i.e., intelligent objects, are *static* and developed according to the domain ontology. They are depicted in Fig. 8: SIBSearch, WebSearch, Navigator (Personal Space), and CarParkService (Smart City Space). The static agents do not exploit the adaptation framework explained in this paper. The adaptation framework based agents are *dynamic* and are presented inside the adaptation framework's execution elements in Fig. 8. All the agents are coded using Python and Qt C++/QML programming language. The same functionality can also be made with other programming languages, e.g., Java, C, C#.

5.2. Adaptation activities

The key to enable changes in dynamic behavior is the ontology published by the domain agents (intelligent objects). The ontology provides a static guideline to be used when designing the dynamic behavior. The domain agent publishes all the properties and functionalities in their ontology and thereby allows the adaptation element to control them. The domain agent may restrict the access to its information or its control in the used smart space, and in such case authentication of the adaptation elements are required.

The adaptation element executes the rules and makes a decision based on the context and quality management information whether to make an adaptation or not. The adaptation uses the quality management's attributes to decide whether this adaptation can be made or not.

By default, the adaptation updates the information to the smart space it is connected to. Therefore, the designer of the actual rules must take this into account and deploy the adaptation into the right smart space. Of course, context information can be added to the rule and update the resulting command into another space, as was described earlier in the context monitoring agent.

Run-time security management is used to manage user authentication in the personal space, i.e., in the smart phone. Authenticated users can also receive more information about an accident if one occurs on the current route and the user has access rights to that information. The security management deals with authentication and access control actions autonomously in this example case. Therefore, the security agents (monitor, analyzer, and planner) are not present in the annotated sequence diagram, Fig. 10. This simplifies the scenario figure. If security-related actions require interaction with other agents, then they have to be presented in the scenario naturally.

Run-time performance management is used for selecting the service that provides better performance. In this case, the age of the information was used as the measurable quality attribute. The web page information is updated at regular intervals, thus it has a fixed maximum value for the age. In case of smart space access, the information is transferred from city smart space to the personal space and age is related to the delay of that transfer.

5.3. Application development steps

The following sections describe the application development with the GuideMe according to the steps described in Fig. 2.

5.3.1. Describe application scenario

After creating short textual description of the application and possible a contextual drawing, such as Fig. 9, start to work with ontologies. First, the application developer has to familiarize herself with the available ontologies, create a new one for her application domain or enhance the existing ones. When all the relevant concepts are available in the ontologies, she starts to sketch the scenarios and related rules. For the illustration she uses Message Sequence Charts (MSCs) as they are typically used in requirement elicitation and functional specifications. The actors in MSCs are SIB(s), the relevant agents from the adaptation framework, and intelligent objects needed for the application.

The MSCs are used to describe both the agent's functionality and the desired behavior in the form of rules, or either one in detail. If the agent has some static parts, it is better to include the rules as one sequence in the chart than to create a single MSC for the rule. In addition, there can be fully dynamic agents, such as the context-reasoning agent, whose actions are controlled by the dynamically changing rules.

The implementation of the rules will be easier and faster, when MSCs are created based on the existing

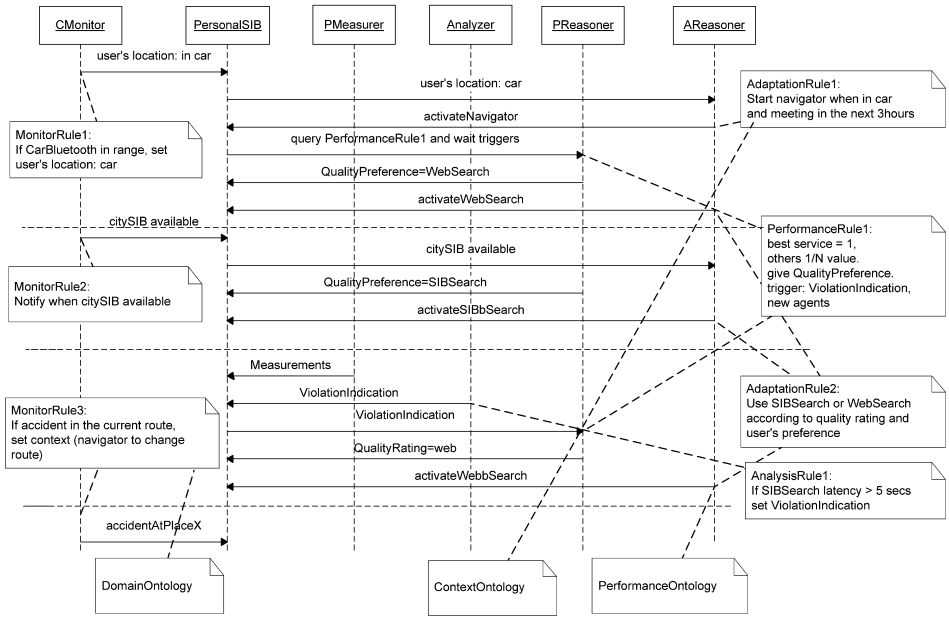

Fig. 10. Overview of GuideMe scenarios as an MSC.

ontologies, since the developer does not have to look for them first. The components defined in the MSCs, i.e., SIBs and agents, must be defined clearly, because the agents can communicate with multiple smart spaces and different smart spaces store information from different sources. After this step, the developer has MSC(s) sketched with actors and can go deeper into the behavior as follows.

5.3.2. Annotate behavior descriptions

In behavior descriptions, the developer follows the guidelines:

- First, present the behavior as analysis models and adaptation rules. The analysis models are either English sentences or if-then-else statements. The adaptation rules are usually given as if-then-else statements.
- Second, describe the basic query and insert actions between the agents and SIBs so that these executions comply with the actions described in the rules part. This way the developer transforming the analysis models into SPARQL queries can check the relevant ontology concepts and target SIBs.

Figure 10 provides an overview of GuideMe scenarios as one MSC. It shows the necessary rules for the different agents. The benefit of showing multiple scenarios in one MSC is that the whole situation of an application can be comprehended at once. The downside is that accurate information, such as the ontology concepts used by the rules, cannot be provided. It is assumed that all the needed information, e.g., Bluetooths discovered and SIBs are available in the SIB used. The context monitor (CMonitor) has a few rules, from which *MonitorRule1* is the one that actually starts this scenario by informing that the user is in the car (i.e., an event). The various context monitors can use rules and be dynamic or monitor something according to the ontology drill-down and thereby be static in a sense. Different types of monitors are discussed and presented in [24]. The AdaptationReasoner (AReasoner) agent has two different adaptation rules assigned to it. *AdaptationRule2* decides which service provider (triggered by *AnalysisRule1*) is used according to the results provided by the PerformanceReasoner (PReasoner) and user preferences.

The MSC in Fig. 11 shows how the adaptation framework handles a situation in which the citySIB is available but the citySIB's performance is not at a

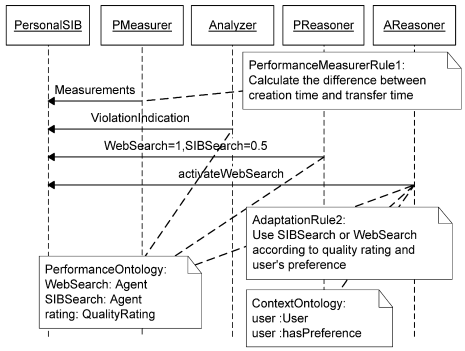

Fig. 11. Changing the information provider according to the context.

satisfactory level and WebSearch is activated again. PMeasurer calculates a value for any information that has properties for creation and transfer time, as shown in *PerformanceMeasurerRule1*. This finer-grained MSC has a more detailed sequence description and it annotates the ontologies used and concepts in the rules, so that the developer can quickly check the correct classes and properties that the rules use. This is specifically useful if the developer does not have previous experience using the specific ontology. Therefore, describe sub-scenarios in more detail, so that the developer can drill down to more detail when needed.

AdaptationRule2 shows the default way for the agent to make a SPARQL subscription from the rule. *PerformanceRule1* exemplifies how the agent is triggered to query and the trigger is a set of triples, such as ViolationIndication. The agents subscribe to the given triples or triple patterns according to the instructions in the SIB.

5.3.3. Transform behavior descriptions to rule language

Next, the developer extracts the rules to the SPARQL queries following the SPARQL features provided by the SIB. This is done by hand, but could be automated if the MSCs contain all relevant information required for the rules.

MonitorRule1, *PerformanceMeasurerRule1*, and *AdaptationRule2* were selected from the MSCs in Figs 10 and 11 to present how the behavior is transformed into the SPARQL queries, and to illustrate three types of rules used in monitoring, reasoning, and adaptation. Figure 12a) shows *MonitorRule1* (Event Trigger), which starts the navigation: If the car's Bluetooth is discovered, inform that the user is in the car. The context ontology (CO4SS) and domain ontology need to be consulted to find the correct concepts to be used in the rule in order to compare the Bluetooth identification description (ID) to the one that is assigned to the user.

Figure 12b) shows the *PerformanceMeasurerRule1* (Calculator) which is used by the PMeasurer agent to measure the latency of the service. This SPARQL query takes all data elements that have a creation time and a transfer time, makes an arithmetic calculation with these values, and returns the results. This rule will be re-run whenever some property of the information changes. Therefore, this rule might be a rather heavy resource user. The effect of the rule can be decreased by adding a FILTER operation to the rule. With the FILTER operation the focus can be set to a specific agent or a specific type of information.

Figure 12c) shows the SPARQL queries associated with *AdaptationRule2* (Context Adaptation), which makes a selection between SIBSearch and WebSearch agents according to the given quality preferences and the user preferences. The developer looks at the ontologies associated with this rule: performance ontology (RPM), context ontology (CO4SS), and domain ontology. From the domain ontology the developer finds out that the provider agents have a preference property which shows the preferences for this agent. In addition, the domain ontology states who has assigned the preference, such as the user or the quality management.

This *AdaptationRule2* requires more than one SPARQL query to cover all the necessary functionalities. The first rule activates the correct provider agent according to the preferred assigner. The assigner can be, for example, the user or quality agent, and is deduced by another rule or defined in the ontology. The second rule, on the other hand, deactivates the rest of the provider agents. These adaptation rules are the most difficult rules to implement, since the developer needs to know how the different execution elements are controlled and possible use concepts from all of the ontologies used to create these rules.

5.3.4. Implement dynamic behavior of intelligent applications

In the last step, the developer codes the agents using Java, C, C#, Python or Qt C++/QML and uses the appropriate KPI library to access SIB(s). She also chooses the relevant SIB implementation that meets her needs. More details, such as performance results,

```
PREFIX :     < http://www.SOFIA.net/Context#>
PREFIX d:      <http://www.SOFIA.net/Domain#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
    :user :location :car .
} WHERE{
     ?endpoint rdf:type d:Bluetooth .
     ?endpoint d:hasID ?id .
     :user d:hasCarBluetoothID ?bluetoothID .
     FILTER(?id = ?bluetoothID) .
}
```

a) SPARQL query of MonitorRule1

```
PREFIX :     < http://www.SOFIA.net/Performance#>
CONSTRUCT {
       ?info :measuredValue ?value .
} WHERE{
       ?info:hasCreationTime ?creationTime .
       ?info:hasTransferTime ?transferTime .
BIND(?transferTime - ?creationTime AS ?value)
}
```

b) SPARQL query of PerformanceMeasurerRule1

```
PREFIX :     <http://www.SOFIA.net/Performance#>
PREFIX c:      <http://www.SOFIA.net/Context#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

# Activate the correct service
CONSTRUCT {
       ?activeAgent :hasStatus "Active" .
  } WHERE{
       c:AdaptationRule2 c:preferAssigner ?a .
       ?activeAgent rdf:type :Agent ;
                        :hasRating ?rating .
       ?rating :assignedBy ?a .
       ?rating :hasValue ?x .
       FILTER(?x = 1) .
}

# Deactivate other services
CONSTRUCT {
       ?activeAgent :hasStatus "InActive" .
 } WHERE{
       c:AdaptationRule2 c:preferAssigner ?a .
      ?activeAgent rdf:type :Agent ;
                    :hasRating ?rating .
       ?rating :assignedBy ?a .
       ?rating :hasValue ?x .
       FILTER(?x < 1) .
}
```

c) SPARQL queries of AdaptationRule2

Fig. 12. a)–c) Differences in the rules.

from the different SIB implementations are available in [51].

As described earlier, the GuideMe is implemented by Python and Qt C++/QML and it uses RIBS [54]. In fact, the SPARQL subscribe feature of RIBS is used to process all the rules (Option #1). That is, the agents make subscriptions according to the rules allocated to them. This reduces communication overhead but adds processing overhead to the RIBS side. This is acceptable, since the subscriptions, i.e., rules, are executed only when something affecting them is changed in the database. Option #2, which can be used in special cases, is to give the rule execution control to the agent. This can be achieved by changing the SPARQL subscriptions to normal queries that the agents make when something changes. Option #3 is to instantiate a local RIBS, make the SPARQL subscriptions to that RIBS, and update all the relevant information to the local RIBS.

The benefit of using CONSTRUCT instead of INSERT, for example, is that the resulting fact can be checked before actually making the adaptation it is going to fire. For example, the adaptation element can check all the necessary qualities before actually executing the command. In addition, one can include privacy and user preferences into the reasoning process.

5.4. Lessons learned in application development

The benefit of using the adaptation framework is that the developer does not have to manually code the adaptation actions inside the agents but can instead define and implement this behavior with rules, i.e., SPARQL queries. Therefore, the dynamic agents, execution elements, and static agents are tested and verified before their usage, and the only variable parts are the GUI (if needed) and the SPARQL queries. This saves time in the application development and enables the developer to focus on the actual logic of the application and overall user experience.

Although application development with the adaptation framework is quite easy (One does not need to code applications with traditional programming languages if not desired), the developer has to have a broad knowledge of SPARQL queries and the ontologies used in order to be able to make application efficiently. If the application development process is divided between different developers, then the developer responsible for the last part, i.e., making

SPARQL queries and adaptations, is the one who should have the most experience, since it combines the earlier development.

5.4.1. Ontologies

In this case study the developer making the adaptation rules did not have prior experience in the performance ontology, and therefore had to rely on the information provided in the MSCs and look for detailed information from the performance (RPM) ontology. The required information was quite easily fetched from the RPM ontology, but some specifics regarding usage of the ontology had to be requested from the RPM ontology developer.

In addition, if the ontologies used are developed parallel to the actual applications, then the changes in ontology might break the SPARQL queries or cause unexpected behavior. Of course, the problem is the same in all ontology-driven development.

5.4.2. Transform behavior descriptions to rule language

It is important to follow the correct concepts from the ontologies used to form the rules so that SPARQL queries can be created without having to contemplate the concepts used. The correctness of concepts is even more important if the SPARQL queries are done by someone other than the developer of the MSCs, which is a common case.

The most error-prone phase in application development is the creation of the SPARQL queries and their management, due to dynamicity of the rules and the cross-effects they can have. Therefore, the developer has to check that the ontology concepts used are correct and that the created SPARQL query is valid. The developer can create the resulting SPARQL query in an iterative way, so that testing the correctness is easier. The developer may perform the necessary verification and validation (V&V) activities according to the inputs and outputs with testing agents, which can insert the needed input values and consume the resulting information.

5.4.3. Results of the case study

The case study implementation showed that the adaptation framework helps in the application development of situation-based smart space applications. The main benefit is that it provides a framework that the developer can exploit in creating dynamic applications that are context-aware and proactively monitor and reason the situation in smart spaces. The framework takes care of handling the rules and allows the behavior to be changed at run time if required.

The SPARQL queries used in implementing the actual dynamism and reasoning were found to be useful in context and security monitoring, reasoning, and adaptation. For performance, it was also easy for monitoring, but for reasoning and adaptation the complexity of the performance-related rules was difficult to manage with SPARQL queries. Note that SPARQL extensions were not yet used.

6. Discussion

In this section, the impact of the proposed adaptation framework to the software development is explored. The presented adaptation framework has been developed and experimented incrementally during the three years in the SOFIA project. Firstly, the evaluation of the adaptation framework is described. Secondly, the summary of experiments is presented. Then, the SUM-SS pilot with the evaluation results is introduced. Thereafter, the evolution of dynamic systems is discussed. Finally, the topics identified for further study are summarized.

6.1. Evaluation steps of the adaptation framework

The IOP principles (cf., Table 2) have guided the work for the framework presented and they have been used as evaluation criteria. The work is also done as guided in the eight lessons dedicated for infrastructure software that supports construction or operation of other software [9]. Next, the work related to these eight lessons is introduced.

Lesson 1. Prioritize core infrastructure features – The fourteen IOP principles are available.

Lesson 2. First, build prototypes that express the core objectives of the infrastructure – this is the approach the SOFIA project had while proceeding toward this adaptation framework. A greenhouse demo was made in the 1st year and that demo was based on existing assets. Thus, the implementation effort was minimized. The greenhouse demo is presented with more detail in [10] and videoed in [47].

Lesson 3. Any test application built to demonstrate infrastructure must also satisfy the criteria of usability and usefulness – run-time security, performance management, and context-awareness were demon-

strated in separated demos in the 2nd year. All are useful on their own. Experiments on context-awareness are reported in [37–39]. The run-time performance management is shown in [10,44]. The run-time security management is exemplified, for example, in [10,14,15].

Lesson 4. Initial proof-of-concept applications should be lightweight – during the 3rd year the SUM-SS pilot [50] was developed in three increments. The pilot was partially tested in different contexts/audiences and with different users.

Lesson 5. Be clear about what your test-application prototypes will tell you about your infrastructure – security and context-awareness were merged first at the conceptual level [13]. This convinced us of the potential for implementation. The way of doing so seemed to be fine in principle but refinements were needed. Therefore, the new framework is presented in this paper.

Lesson 6. Do not confuse the design and testing of the experimental infrastructure with the provision of an infrastructure for experimental application developers – the work was kept within a core research group (6 persons).

Lesson 7. Define a limited scope for test-applications and permissible uses of infrastructure – This was done by delivering a part of our SUM-SS pilot infrastructure to be integrated with the Smart Maintenance pilot [50]. The SUM-SS pilot was tested by local and remote end-users, of whom 80% were not involved in the SOFIA project in any form (see Fig. 14).

Lesson 8. There is no point in faking components and data if you intend to test for user experience benefits – the SUM-SS pilot [50] was complex enough for evaluation purpose and valuable feedback was also received from end-users with open questions. However, their presentation is beyond the scope of this paper due to space limitations.

6.2. The summary of experiments before the pilot

When comparing our work related to the eight lessons above, some of our published results are referenced. Table 3 summarizes the experiments performed and knowledge achieved from the demonstrations and the SUM-SS pilot. Links to the video demonstrations are also provided.

6.3. The SUM-SS pilot

The SUM-SS [50] pilot integrates the aforementioned separate demonstrations. Thus, the GuideMe application, used as an example application in this paper, is also part of it. The objective of the SUM-SS [50] pilot is to demonstrate seamless usage of multiple smart spaces, including a personal space, a smart home, smart office, and smart city. The first three spaces collaborate with the services provided in a cloud through the Cam4Home Open Platform [4]. The setup of the SUM-SS is illustrated in Fig. 13.

The SUM-SS pilot was evaluated by 28 end-users. Three evaluations were done during exhibitions at the ARTEMIS Technology Conference 2011 in Bologna and at the ARTEMIS/ITEA2 Co-Summit 2011 in Helsinki. The rest of the evaluations were performed in our demonstration environment, the Innovation Kitchen, at VTT Oulu's premises. As mentioned above, most of the evaluators (80%) were not involved in the SOFIA project in any form.

The estimated time for each evaluation effort was fixed at a maximum of two hours. The evaluations were made in nine groups and each group had 2–4 evaluators at the same time in the same place. The number of evaluators in a group was fixed so that they could test the features at the same time without any waiting periods. The evaluators were asked to fill the Web questionnaire after the evaluation and during the same day. Thus, the evaluation results and primary reactions to the smart space applications they had tested could be kept objective. As justification, part of the evaluation results is depicted in Fig. 14. The factors illustrated are:

1. All capabilities the user expects are supported.
2. A feeling of being in control is provided.
3. The user's task is assisted and made easier.
4. It is simple to use and easy to learn.
5. The experience is effortless and seamless.
6. It is easy to recover whenever a mistake is made.
7. It is effective in helping the user to complete the tasks.
8. The user is able to work with it efficiently.
9. It helps the user to save time.
10. The user feels comfortable using it.

As can be seen, the end-users (the evaluators) were well satisfied with the functional capabilities of the SUM-SS pilot. Factors related to users' experiences (2, 4, 9 and 10) are especially high (over 3/max scale 4). This can be seen as a strong indicator of the right research direction and results received so far.

Table 3

Demonstrations with specific validation focus

Case	Description of the case
Smart Greenhouse demo [10] Smart Greenhouse video [47]	The demo illustrated: i) how different devices communicate and semantically understand the information exchanged, ii) how changed quality requirements are dynamically fulfilled by context-based self-adaptation, and iii) how the smart space evolves and extends incrementally without modification to the existing devices.
Fictitious scenario of an emergency scenario in a smart city; feasibility of the context ontology [40]	The context-awareness concept, predecessor of the CAMA, was validated against a fictitious scenario. The predecessor of the CO4SS was analyzed to be sure of its quality as a context specification to fulfill the requirements [42] set for the context definition.
Cross-domain (personal space and person's smart home) scenario to wake the person up [37]	The context-awareness agents were created based on the CAMA with design-time configuration (Within the adaptation framework it is also possible to do the configuration at run time). The required functionality was activated according to the rules and existing situation(s). The rules and ontology were configured with a Smart Modeller Tool [20,21].
Lighting scenario [38]	The CO4SS was used at development time to add the necessary functionality for both the personal and home space context-awareness agents. The CO4SS was expandable with domain specific parts that were created to control the lighting in the home.
Maintenance process of a smart building [39]	To map context and domain-specific ontologies, i.e., the CO4SS and the existing maintenance ontology were aligned to share context to supervise the progress of the maintenance scenario. The Context Monitor agent showed to be reusable and reconfigurable at run time.
Risk-based security adaptation in a greenhouse [10,11]	The greenhouse with a shopping area constitutes a public smart space. In the smart space, threats increase risk levels and security mechanisms decrease risks. Hence, the monitoring concentrates on recognizing threats. In this case confidentiality and integrity security attributes are considered. Furthermore, user authentication is performed by means of gait information stored via acceleration sensors inside the mobile phone.
Adaptive user authentication [14]	The first case that utilizes knowledge from the ISMO. It adapts user authentication by monitoring authentication related measures: password length, age, variation of characters, and session duration. Important information is available only when an acceptable authentication level is reached. User re-authentication is requested when the session duration is long.
Adaptive user authentication [54]. This is related work from our colleagues and done also in the SOFIA project.	Change protection access control policies according to the user's role or popularity of information. The 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 [12]	The paper and related case example shows how easily knowledge in the ISMO can be extended. Moreover, design steps to develop adaptable security were presented.
Adaptation based on measured performance at run time [44]	Foundational work to the run-time performance management of information-broker based adaptive applications. The experiment with adaptation based on resource requirements is done in the smart greenhouse demo [10].
Reusable agent for smart environments [24]	A reusable design for monitor managers and monitors is introduced and validated in a smart space laboratory case. This work illustrates how reusable agents can be designed and implemented, i.e., how to apply *for reuse* and *with reuse* practices in the development of smart environments.
Smart door video [46]	The home smart space contains both private and public areas. The smart space enforces different access controls for local and remote usage, and uses contextual information (criticality of actions).

6.4. Future work

At the moment, the adaptation takes into account who has assigned a rule for it to execute, and therefore pays attention to users' security. The security features, such as user authentication and access control, do exist for enabling rule execution according to user permissions. In the future, more quality attributes such as reliability will be added to this adaptation framework. Support for the trade-off between different quality attributes will be also added. Further investigation is needed to find out how complicated analysis models can be used without effects on the overall performance of the situation-based and self-adaptive application.

The creation of dynamic models, the analysis models, and the adaptation rules are laborious work. A tool for this will be developed. A testing tool is also planned to be developed for being able to find the collision between the rules and quality attributes so that the circular effects can be hindered.

To ease the development of performance-related rules it is intended to implement some specific extension functions for the SPARQL 1.1 reasoning engine

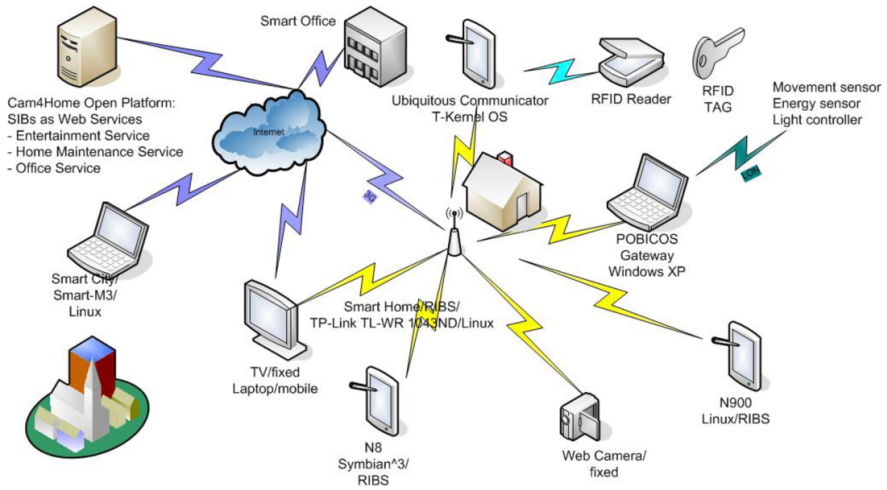

Fig. 13. Setup for the SUM-SS.

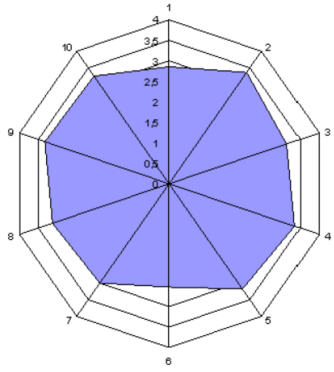

Fig. 14. Maturity of functional capabilities of the pilot.

used (implemented as part of RIBS). This is required because performance rules are by nature more complicated, due to the inherent complexity of the calculations in comparison with the context-related rules.

7. Conclusions

This paper introduced an adaptation framework for the situation-based and self-adaptive applications of smart environments. The framework has a flexible and reusable architecture based on a set of micro-architectures that follow the MAPE-K pattern. The evolution of the framework is based on generic ontologies for context, security, and performance management. The execution of the framework is supported by the dynamic models for performing run-time reasoning and adaptation. The framework maximizes reuse due to: i) the separation of generic and domain knowledge, ii) the standardized way of modeling knowledge and rules, and iii) all ontologies or parts of them can be connected to the situation-based application via the rules.

The presented application framework is intended for an application developer who is i) creating an application scenario, and ii) transforming the scenario to annotated sequence diagrams with the help of the ontologies and related rules. Thereafter, the application developer iii) transforms the rule definitions to the SPARQL queries.

Our approach was exemplified through creation of the GuideMe application, which exploits context, security, and performance information for adapting the service according to the quality requirements and the context of the user, as well the smart environment, without bothering the end-user. The implemented GuideMe is able to identify situations and adapt itself accordingly.

Validation of the adaptation framework was made incrementally: Eight evaluation steps were performed resulting in twelve demonstrations and a cross-domain pilot that integrates smart environments with cloud services and provides seamless usage of situation-based and self-adaptive applications for users of personal spaces, smart homes, smart offices, and smart cities.

References

[1] D. Alur, J. Crupi and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, Sun Microsystems Press, 2nd edn, 2003.

[2] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Niclas, A. Ranganathan and D. Riboni, A survey of context modelling and reasoning techniques, *Pervasive Mob. Comput.* **6**(2) (April 2010), 161–180.

[3] M. Bezold and W. Minker, A framework for adapting interactive systems to user behavior, *Journal of Ambient Intelligence and Smart Environments* **2**(4) (April 2010), 369–387, IOS Press.

[4] Cam4Home Project, www.cam4home-itea.org, June 2012.

[5] H. Chen, H.T. Finin and A. Joshi, *The SOUPA Ontology for Pervasive Computing*, Whitestein Series in Software Agent Technologies, Springer, 2005.

[6] D.J. Cook and S.K. Das, How smart are our environments? An updated look at the state of the art, *Pervasive Mob. Comput.* **3**(2) (March 2007), 53–73.

[7] L. Daniele, P. Dockhorn Costa and L. Ferreira Pires, Towards a rule-based approach for context-aware applications, in: *Dependable and Adaptable Networks and Services*, A. Pras and M. van Sinderen, eds, EUNICE 2007, Vol. 4606, Springer-Verlag, Berlin and Heidelberg, 2007, pp. 33–43.

[8] G. Dobson, R. Lock and I. Sommerville, QoSOnt: A Qos ontology for service-centric systems, in: *Proc. of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, IEEE Computer Society, 2005, pp. 80–87.

[9] W.K. Edwards, V. Bellotti, A.K. Dey and M.W. Newman, Stuck in the middle: The challenges of user-centered design and evaluation for infrastructure, *Paper/Short Talks: Issues in Software Development, CHI 2003: New Horizons* **5**(1) (2003), 297–304.

[10] A. Evesti, M. Eteläperä, J. Kiljander, J. Kuusijärvi, A.Purhonen and S. Stenudd, Semantic information interoperability in smart spaces, in: *Proc. of the 8th Int. Conference on Mobile and Ubiquitous Multimedia (MUM'09)*, ACM International Conference Proceedings Series, 2009, pp. 158–159.

[11] A. Evesti and E. Ovaska, Ontology-based security adaptation at run-time, in: *Proc. of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, IEEE Computer Society Washington, DC, USA, 2010, pp. 204–212.

[12] A. Evesti and E. Ovaska, Design time reliability predictions for supporting runtime security measuring and adaptation, in: *Proc. of the 3rd Int. Conference on Emerging Network Intelligence (EMERGING'11)*, IARIA, 2011, pp. 94–99.

[13] A. Evesti and S. Pantsar-Syväniemi, Towards micro architecture for security adaptation, in: *Proc. of the 4th European Conference on Software Architecture, ECSA 2010 Workshops: 1st Int. Workshop on Measurability of Security in Software Architectures (MeSSA 2010),* ACM, New York, NY, USA, 2010, pp. 181–188.

[14] A. Evesti, R. Savola, E. Ovaska and J. Kuusijärvi, The design, instantiation, and usage of information security measuring ontology, in: *Proc. of the 2nd Int. Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS 2011),* IARIA, 2011, pp. 1–9.

[15] A. Evesti, J. Suomalainen and E. Ovaska, Reusable security adaptation approach for smart spaces, *Pervasive and Mobile Computing*, submitted.

[16] S. Ferrari, *Gestione Del Contesto Su Architettura Smart M3*, Tesi di Laurea in Calcolatori Elettronici L-A (Sessione III, Anno Accademico 2009/10), University of Bologna, Italy, 2011.

[17] F. Garcia, F. Ruiz, C. Calero, M.F. Bertoa, A. Vallecillo, B. Mora and M. Piattini, Effective use of ontologies in software measurement, *Knowl. Eng. Rev.* **24**(1) (March 2009), 23–40.

[18] K. Henricksen, J. Indulska and A. Rakotonirainy, Using context and preferences to implement self-adapting pervasive computing applications, *Software Pract. Exper.* **36**(11–12) (September 2006), 1307–1330.

[19] J. Honkola, H. Laine, R. Brown and O. Tyrkkö, Smart-M3 interoperability platform, in: *Proc. of the 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010) in IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society, 2010, pp. 1041–1046.

[20] A. Katasonov, Enabling non-programmers to develop smart environment applications, in: *Proc. of the 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010) in IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society, 2010, pp. 1059–1064.

[21] A. Katasonov and M. Palviainen, Towards ontology-driven development of applications for smart environments, in: *Proc. of the 8th PERCOM Workshops*, IEEE Press, 2010, pp. 696–701.

[22] J.O. Kephart and D.M. Chess, The vision of autonomic computing, *Computer* **36**(1) (January 2003), 41–50.

[23] A. Kosek, A. Syed and J. Kerridgey, RDF recipes for context-aware interoperability in pervasive systems, in: *Proc. of the 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010) in IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society, 2010, pp. 1004–1009.

[24] J. Kuusijärvi and S. Stenudd, Developing reusable knowledge processors in smart environments, in: *Proc. of the 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT2011), the 2nd Int. Workshop on Semantic Interoperability for Smart Space (SISS 2011)*, IEEE Computer Society, pp. 286–291.

[25] I. Lera, C. Juiz and R. Puigjaner, Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems, *Science of Computer Programming* **61**(1) (June 2006), 27–37.

[26] H.H. Lund, T. Klibo and C. Jessen, Playware technology for physically activating play, *Artificial Life Robotics* **9**(4) (2005), 165–174.

[27] V. Luukkala and I. Niemelä, Enhancing a smart space with answer set programming, in: *Semantic Web Rules,* M. Dean, J. Hall, A. Rotolo, and S. Tabet, eds, RuleML 2010, Vol. 6403, Springer-Verlag, Berlin and Heidelberg, 2010, pp. 89–103.

[28] M.A. De Miguel and M.T. Higuera, Runtime management of quality specification for QoS-aware components, in: *Proc. of the 30th EUROMICRO Conference*, IEEE Computer Society Washington, DC, USA, 2004, pp. 84–91.

[29] S.B. Mokhtar, N. Georgantas and V. Issarny, COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support, *Journal of Systems and Software* **12**(80) (December 2007), 1941–1955.

[30] K. Nahrstedt, D. Xu, D. Wichadakul and B. Li, QoS-aware middleware for ubiquitous and heterogeneous environments, *IEEE Communications Magazine* **39**(11) (November 2001), 140–148.

[31] E. Niemelä, J. Kalaoja and P. Lago, Toward an architectural knowledge base for wireless service engineering, *IEEE Transactions on Software Engineering* **31**(5) (May 2005), 361–379.

[32] E. Ovaska, T. Salmon Cinotti and A. Toninelli, Design principles and practices of interoperable smart spaces, in: *Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools*, X. Liu and Y. Li, eds, IGI Global, 2012, pp. 18–47.

[33] 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) (June 2010), 577–601.

[34] P. Oreizy, N. Medvidovic and R.N. Taylor, Runtime software adaptation: Frameworks, approaches and styles, in: *Proc. of the 30th International Conference on Software Engineering, ICSE 2008*, ACM, New York, NY, USA, 2008, pp. 899–909.

[35] OSGi Alliance, OSGi Platform – Service Compendium, Technical Report Release 4, Version 4.1, OSGi, 2007.

[36] OWL, Web ontology language, www.w3.org/2004/OWL, June 2012.

[37] S. Pantsar-Syväniemi, J. Kuusijärvi and E. Ovaska, Context-awareness micro-architecture for smart spaces, in: *Advanced in Grid and Pervasive Computing*, J. Riekki, M. Ylianttila and M. Guo, eds, GPC 2011, Vol. 6646, Springer-Verlag, Berlin and Heidelberg, 2011, pp. 148–157.

[38] S. Pantsar-Syväniemi, J. Kuusijärvi and E. Ovaska, Supporting situation-awareness in smart spaces, in: *Grid and Pervasive Computing Workshops*, M. Rautiainen, T. Korhonen, E. Mutafungwa, E. Ovaska, A. Katasonov, A. Evesti, H. Ailisto, A. Quigley, J. Häkkilä, N. Milic-Frayling and J. Riekki, eds, Vol. 7096, Springer-Verlag, Berlin and Heidelberg, 2012, pp. 14–23.

[39] S. Pantsar-Syväniemi, E. Ovaska, S. Ferrari, T. Salmon Cinotti, G. Zamagni, L. Roffia, S. Mattarozzi and V. Nannini, Case study: Context-aware supervision of a smart maintenance process, in: *Proc of the 11th IEEE/IPSJ Int. Symposium on Applications and the Internet (SAINT2011), the 2nd Int. Workshop on Semantic Interoperability for Smart Space (SISS 2011)*, IEEE Computer Society, 2011, pp. 309–314.

[40] S. Pantsar-Syväniemi, K. Simula and E. Ovaska, Context-awareness in smart spaces, in: *Proc. of the 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010) in IEEE Symposium on Computers and Communications (ISCC 2010)*, IEEE Computer Society, 2010, pp. 1023–1028.

[41] V. Peristeras and K. Tarabanis, The connection, communication, consolidation, collaboration interoperability framework (C4IF) for information systems interoperability, *IBIS – Interoperability in Business Information Systems* **1**(1) (March 2006), 61–72.

[42] D. Preuveneers and Y. Berbers, Internet of things: A context-awareness perspective, in: *The Internet of Things: From RFID to the Next Generation Pervasive Networked Systems*, L. Yan et al., eds, Auerbach Publications, 2008, pp. 287–307.

[43] D. Preuveneers and P. Novais, A survey of software engineering practices for the development of smart applications in Ambient Intelligence, *Journal of Ambient Intelligence and Smart Environments* **4**(3) (June 2012), 149–162, IOS Press.

[44] A. Purhonen and S. Stenudd, Runtime performance management of information broker-based adaptive applications, in: *Software Architecture*, I. Crnkovic, V. Gruhn and M. Book, eds, ECSA 2011, Vol. 6903, Springer-Verlag, Berlin and Heidelberg, 2011, pp. 203–206.

[45] M. Salehie and L. Tahvildari, Self-adaptive software: Landscape and research challenges, *ACM Transactions on Autonomous and Adaptive Systems* **4**(2) (May 2009), Article No. 14.

[46] Smart door video, www.youtube.com/watch?v=anRW0y2r1Q0, June 2012.

[47] Smart Greenhouse video, www.youtube.com/watch?v=EU9alk9t7dA, June 2012.

[48] Smart-M3 open source project, sourceforge.net/projects/smart-m3, June 2012.

[49] Smodels, www.tcs.hut.fi/Software/smodels/, June 2012.

[50] SOFIA pilots, includes SUM-SS and smart maintenance pilot descriptions, www.sofia-community.org/files/brochures_and_presentations/SOFIA_Pilots_&_Community_Brochure.pdf, June 2012.

[51] SOFIA reference implementations and their evaluation results, www.sofia-community.org/files/D5.42_Reference_Implementation_of_the_Interoperability_Platform.pdf, June 2012.

[52] A. Soylu, P. De Causmaecker and P. Desmet, Context and adaptivity in pervasive computing environments: Links with software engineering and ontological engineering, *J. of Software* **4**(9) (November 2009), 992–1013.

[53] SPARQL query language for RDF, W3C recommendation, www.w3.org/TR/rdf-sparql-query/, June 2012.

[54] J. Suomalainen, P. Hyttinen and P. Tarvainen, Secure information sharing between heterogeneous embedded devices, in: *Proc. of the 4th European Conference on Software Architecture, ECSA 2010 workshops: 1st Int. Workshop on Measurability of Security in Software Architectures (MeSSA 2010)*, ACM, New York, NY, USA, 2010, pp. 205–212.

[55] T. Strang, C. Linnhoff-Popien and K. Frank, CoOL: A context ontology language to enable contextual interoperability, in: *Distributed Applications and Interoperable Systems*, J.-B. Stefani, I. Demeure and D. Hagimont, eds, DAIS 2003, Vol. 2893, Springer-Verlag, 2003, pp. 236–247.

[56] R.N. Taylor, N. Medvidovic and P. Oreizy, Architectural styles for runtime software adaptation, in: *Software Architecture, 2009 & European Conference on Software Architecture, WICSA/ECSA 2009, Joint Working IEEE/IFIP Conference on*, September 2009, pp. 171–180.

[57] H. Truong, S. Dustdar and T. Fahringer, Performance metrics and ontologies for grid workflows, *Future Gener. Comput. Syst.* **23**(6) (July 2007), 760–772.

[58] X.H. Wang, D.Q. Zhang, T. Gu and H.K. Pung, Ontology based context modeling and reasoning using OWL, in: *Proc. of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society Washington, DC, USA, 2004, pp. 18–22.

[59] W3C, RDF Vocabulary description language, RDF-Schema, www.w3.org/TR/rdf-schema/#ch_domain, June 2012.

[60] W. Zhang, K.M. Hansen and T. Kunz, Enhancing intelligence and dependability of a product line enabled pervasive middleware, *Pervasive Mob. Comput.* **6**(2) (April 2010), 198–217.

PUBLICATION V

Context-awareness in smart spaces

In: Computers and Communications (ISCC),
2010 IEEE Symposium, 22–25 June 2010.
Pp. 1023–1028.

Context-Awareness in Smart Spaces

Susanna Pantsar-Syväniemi
Software Architectures and Platforms, VTT
Oulu, Finland
e-mail: Susanna.Pantsar-Syvaniemi@vtt.fi

Kirsti Simula
Services, Connect
Nokia
Oulu, Finland
e-mail: Kirsti.Simula@nokia.com

Eila Ovaska
Software Architectures and Platforms, VTT
Oulu, Finland
e-mail: Eila.Ovaska@vtt.fi

***Abstract*—Any of our living environments can form a smart space that provides services and applications for the users according to their situations, the computing and communication facilities of the environment and the surroundings of the user. The ability to take into account the context of the user, and the digital and physical environment makes a space smart. This paper introduces a set of new capabilities that are required for achieving context-awareness in smart spaces. First, sensor data from the environment is acquired and aggregated. Second, the context data is interpreted and represented and possibly enhanced with external information. Thereafter, context reasoning is possible. Finally, situation based context reasoning is made by application agents that exploit domain ontologies for tuning context information to the situation in hand. The use of the introduced context-awareness concept is exemplified by an emergency scenario of a smart city.**

Keywords: ontology, context awareness, smart space

I. INTRODUCTION

People's every day life can be enriched by smart spaces that are able to exploit pervasive computing environments embedded into our surroundings. These environments heavily rely on a multitude of sensors and sensor networks that produce a large amount of data to be analyzed and reacted by the user or/and the systems/devices of the space. [1] Smart spaces may have a large amount of different kinds of systems, devices, and sensors that interact with each others in different way; communication may be periodic or ad-hoc over wired and/or wireless networks. Computers also differ in their computing capabilities, such as CPU, processing power, amount of memory and an operating system. Moreover, the internal state of the systems, devices and communication channels is changing during applications operation. This changing state of the execution environment forms the first dimension of context.

User preferences and needs drive the second dimension of context data. User preferences define the user related configuration parameters for applications. What, by whom and how these parameters are changed, is predefined. These configuration parameters can be changed, when user's needs are changing. The change is made by the user itself or by an intelligent agent that observes that user's desire has changed. In the latter case, the intelligence is realized by, e.g. a reinforcement learning service. Challenges of this kind of service include time-dependency of actions and possibly infinite time horizon. Actions available in the same state can be different in different steps of time. In real-world applications, action-state spaces can also be overwhelmingly huge, and therefore, in practice some probabilistic component, which estimates the expected utility of selected action, is needed. Reinforcement learning can be applied to proactively adapting the execution platform for stress peaks caused by users, overwhelming data or/and increased attacks [2].

The third dimension of context is derived from the gathered information about who else can use or is using the space. The content of this dimension depends on which kind of a smart space the context awareness is to be realised. A personal space has to deal information with high privacy. A smart city is intended for offering information services for different types of users, e.g. citizens, tourists, professional users in ordinary and emergency situations. Thus, the role of the space user has a heavy influence on what and how information/service is to be provided. Therefore, this context dimension is very much related to the quality requirements set for the (information) service and its delivery. Typically, they cover the execution qualities, such as security, safety, reliability, availability and performance. Thus, context-awareness is needed, not only for adapting according to resources and preferences but also for managing, perhaps conflicting, quality demands and quality of service agreements the smart space users agree with smart space providers.

The first two dimensions of context data are quite well covered in the literature but the third dimension, the derived context data, is not well defined nor well understood. Thus, our aim is to shed light, especially, on that issue. Due to our interest, i.e. context based quality management at run-time, we approach the question through conceptualization of the elements of the physical, digital and situational context data. The main contributions of this paper are 1) the introduction of a novel context ontology that exploits some parts from SOUPA [3] and the upper context conceptualization presented in [4]; 2) a concept of context-awareness that takes care of context monitoring, context reasoning and context based adaptation; 3) an example scenario used for validating the proposed context-awareness concept within the Smart-M3 architecture [5].

The structure of the paper is as follows. Section 2 presents the background. Section 3 introduces our context ontology and the mechanisms created for handling context

978-1-4244-7755-5/10/$26.00 ©2010 IEEE

information at three abstraction levels. Thereafter, section 4 illustrates how these mechanisms are applied in the emergency scenario happened in Smart City. Conclusion and future work close the paper.

II. BACKGROUND

There is a well known definition for context: 'Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and application, including the user and applications themselves.'[6]

Bettini et al. [7] define three context dimensions; physical context, computational context, and user context. These context dimensions are similar to ours. A context specification is considered a kind of interoperability that sets the following requirements for context definition [8]:

- Context specification must have comprehensive domain coverage and terminology.
- Context specification must be expressive and without semantic ambiguity.
- Context specification shall be processed without complexity.
- Context specification shall be evolvable.

These requirements will be kept as evaluation criteria for the context ontology to be proposed in this paper.

In [7], three approaches of context modeling and reasoning have been analyzed; an object-role based model, a spatial model; and an ontology-based model:

- The strength of the object-role based approach is its support for various stages of the software engineering process. However, its weakness is a 'flat' information model, i.e. all context types are represented as atomic facts.
- The spatial context models are well suited for context-aware applications that are mainly location-based, like many mobile applications. The main consideration of the spatial context model is the choice of the underlying location model. Relational location models are easier to build up than geographic location models because they offer simple mapping to map data and GPS sensor data. The drawback is the effort the special context model takes to gather and keep up to date the location data of the context information.
- Ontological models of context provide clear advantage both in terms of heterogeneity and interoperability. User-friendly graphical tools make the design of ontological context models viable to developers that are not particularly familiar with description logics. However, there is very little support for modeling temporal aspects in ontologies. The main problem is that reasoning with e.g. OWL-DL poses serious performance issues.

In spite of the weaknesses of OWL (Web Ontology Language) we selected it for describing the context ontology due to its support for interoperability and heterogeneity, both important for evolvable ontology.

Kapitsaki et al. [9] classify context-aware service engineering into two classes; language based approaches and model-driven approaches. Language based approaches such as context-oriented programming and aspect-oriented programming follow the separation of concerns; applications are kept context-free and context is handled as a first-class entity of the programming language while separate constructs are used to inject context-related behavior into the adaptable skeleton. Context-aware aspects programming is one step further; the aspects are driven by context, i.e. a particular aspect may or may not be executed depending on the context of use [10]. Due to the fact that Unified Modeling Language (UML) is the most widely accepted modeling language, also several model-driven approaches have emerged [9], [11]. Typically, these approaches introduce a metal-model enriched with context related artifacts, in order to support context-aware service engineering.

III. CONTEXT AWARENESS CONCEPT

Our context awareness concept consists of novel context ontology and agents for context monitoring, reasoning and context-based adaptation. The concept is based on the semantic context information triangle presented by Bettini et al. in Figure 2 [7]. Our context awareness concept is also based on the context-aware middleware architecture of smart spaces introduced in [12]. The context-aware middleware extends the interoperability platform (IOP) so that beside of the context storage, retrieval and distribution via semantic information brokers (SIBs) there are enhanced functionalities available for context specification, acquisition, monitoring, pre-processing, aggregation, reasoning and context-based filtering and context-aware adaptation.

A. Context ontology

As stated in [4], it is not possible to predefine all the dimensions of context, but an application has still to be able to modify its behaviors with respect to this piece of information and its relation with other context dimensions, and thereby improve user experience or behavior of other applications. Adaptability should not be understood as a one-to-one relation between user and application, but in pervasive computing settings it should be considered as a relation between the application and other elements of the settings (e.g. devices, physical environment, users etc.) [4].

The physical context of environment as such can be meaningless for applications. The context reasoning and interpretation is used to provide the context data in a format that the applications understand it. This means that the low-level context information abstracted by creating a new level that gets the sensor perceptions as input and generates or triggers system actions. This higher-level context information is often referred as a situational context. Adaptations in context-aware applications are then caused by the change of situations (i.e., a change of a context value triggers adaptation if the context update changes the situation) [7].

The context information (from the physical context of environment to the digital and situation context) needs to be

presented in a machine understandable format, often also in a human understandable format. We have created our novel context ontology using OWL by importing some parts from the SOUPA [3] 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 [4]. Our context ontology, presented in Figure 1, takes account the different levels of semantic context interpretation and abstraction presented in [7]. The first level, the physical context of environment, detects individual actions like 'Set status as Normal'. The second level, the digital context of environment, fuses the individual output of the first layer. The output of the second layer is a group of situations like 'Accident nearby'. The third level fuses the individual output of the second level and describes the relationships between situations.

In addition, our context ontology, relevant for the smart spaces, has three dimensions: physical, digital and human. These context dimensions are located in Figure 1 to that level they mostly "belong" even if the dimensions can spread out to all three levels. The human dimension is illustrated as user context in the third level even some parts of that dimension like 'the role of smart space user' is relevant in the second level. Our context specific agents are illustrated in Figure 1, in the corresponding interpretation and abstraction levels.

Beliefs of the agent represent the informational state of the agent. The beliefs are usually low level context information and thereby placed in the first level, physical context of environment, where the low level context information is gathered and processed. Desires represent the motivational state of the agent and thereby placed in the second level, digital context of environment. Intends represent the deliberative state of the agent, to which the agent has to some extent committed. The intends are in the third level, situation context, where the agent sets the situation in the environment. The agent has the same meaning as a knowledge processor in [5], [12].

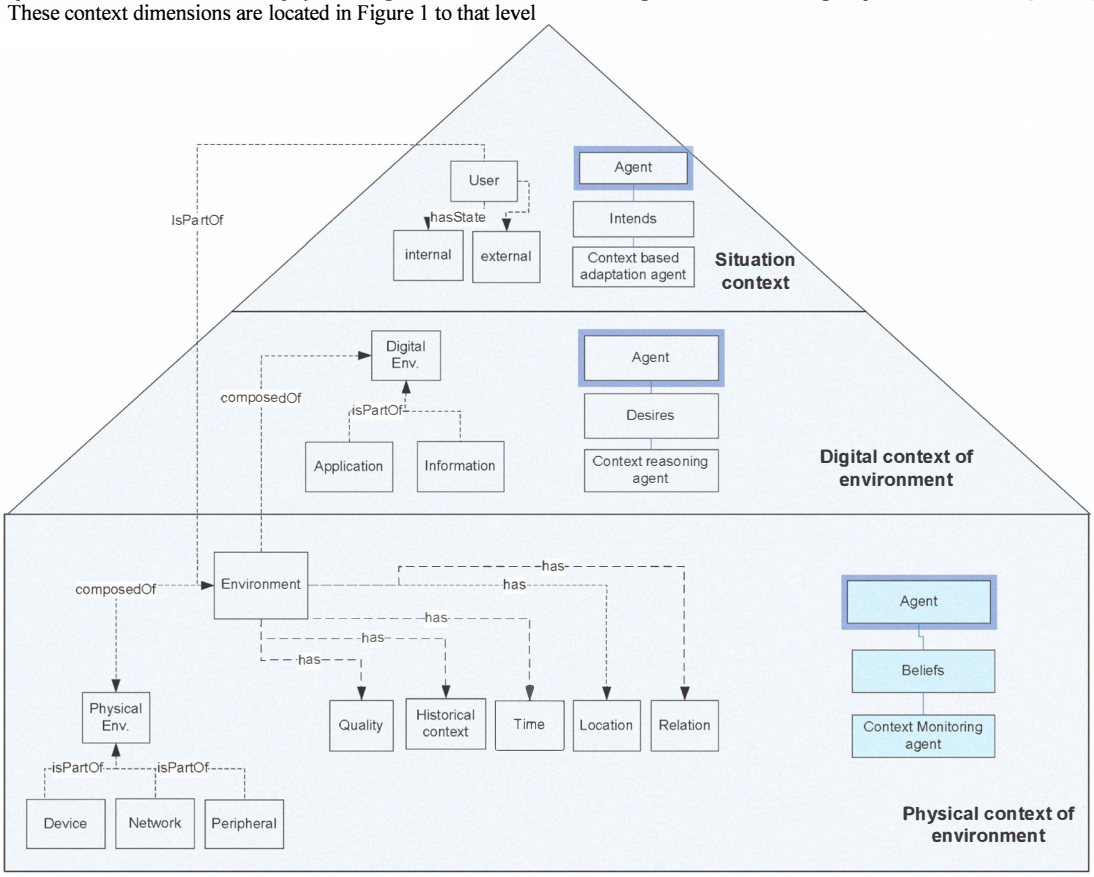

Figure 1. A snapshot of the context ontology

B. Context monitoring

A *context monitoring* agent is used by a smart space application to follow up its scope of the context information. In start-up phase, the used scope is set at design time by the application developer. At run-time the scope can be changed by the application that is in charge or has created this context monitoring agent. The scope is a set of information, e.g. a threshold of battery level, to be followed up by the context monitoring. Thus, the context monitoring agent subscribes to the relevant information located in SIB and gets notifications when information changes. Instead of the subscription, the context monitoring agent can be "polling" the information by querying it from the SIB in certain time period or querying one at a time. The context monitoring agent does not need to be located in the same computing environment than the application since the derived context can be mediated via SIB by using a suitable communication channel between the SIB and the context monitoring agent and between the SIB and the application. The context monitoring is used for measuring current contexts relevant for the smart space application. It is also capable to analyze the validity and quality of context information it receives as a result from the SIB based on the subscriptions or the queries.

C. Context reasoning

A *context reasoning* agent is used by a smart space application to reason based on the context information received. The context reasoning agent is divided to the analyzing and reasoning parts when reasoning is complex, e.g. for reasoning context based on a set of measured quality attributes and their relationships. The basis for reasoning comes from the requirements set. The requirements can be set on design time or they can be given by the user/application on the activation time. Thus, the context reasoning agent, i.e. its controlling rules, is configurable. In the smart space we can have many variations of the context reasoning, e.g. one context reasoning agent for managing one quality attribute. Reasoning agents may use additional context information, e.g. Global Time, Location Based System and Geographical Information System, to enhance the context data of the digital environment with external context data of the space. To reason over (manage) all context reasoning agents, e.g. a set of quality attribute specific reasoning agents, in the smart space we need to have a facility – like a context based adaptation agent.

D. Context based adaptation

A *context based adaptation* agent is used for adapting the smart space applications based on 1) the current context or 2) both the current and the history context. The current context is a snapshot of the smart space at issue, i.e. a situation as described in [13] and in [7]. The context based adaptation agent needs strategies that can be expressed by a means of high-level directives, such as policies and adaptation algorithms that might include a learning part when proactive behavior is needed. In simple cases, adaptation can be processed as part of reasoning agents. However, the agents have to be modular so that reasoning, learning and adaptation algorithms can be changed at run time.

IV. VALIDATION

A. Scenario Description

"Susanna is usually driving by her smart car to her working place. This morning she sees on the display of her car that there is emergency situation in the city on her direction of travel. From the map view she can check the location of the emergency- it is on the main bridge on the highway. She follows the instructions given by navigator to use another way to pass the jammed side of the highway."

The emergency is noticed by smart city authority system(s) based on the contextual information gathered and interpreted. The physical, environmental devices, e.g. cameras located in and near by the bridge, provide information in real time. By utilizing, e.g. the status information of the radio network and permits given by Susanna, the authority with the help of the mobile network operator can provide her the notification before she starts her daily trip to the working place. In the case of emergency the authorities would like to inform people who are 1) already in emergency area, 2) approaching to the emergency area, or 3) starting their daily trip via the emergency area.

In this fictitious scenario we can identify the abovementioned context levels with dimensions (physical, digital, and human). The environmental devices in the city, e.g. cameras in the highway, are providing the bottom level context information based on the ontology for the physical context of environment. The information about the emergency can be reused for different purposes, e.g. to guide rescue forces and to guide people. Thus, that information is modeled by using the ontology for the digital context of the environment. The ontology for the situation context is utilized when Susanna is informed about the emergency while she is starting her daily trip to the workplace.

B. Feasibility of the context ontology

As stated before we use the following criteria for analyzing the quality of the context ontology:

- Coverage and terminology: common concepts and properties of context data are covered. Terminology is defined based on the up to date terminology defined by literature and enhanced with history context and quality of context.
- Expressive and unambiguous specification; the ontology includes three dimensions: physical, digital and human. Environment context is enhanced with time, location, quality, history and relationships. OWL was used as representation language because it is expressive and widely adopted in context and ontology modeling.
- Uncomplicated processing; the context specification created with OWL is both machine and human readable and thereby easy to process. The rules, created for our Smart City scenario, were easy to form based on our context ontology.
- Evolvability; the context specification seems to be easy to extend when using OWL based ontologies. This will be studied in future research activities.

978-1-4244-7755-5/10/$26.00 ©2010 IEEE

However, the real benefit of using ontology for context information in smart spaces, which lies in the interoperability of different devices, will not become effective before there is a widely-accepted standard context ontology.

C. *Use of context specific agents*

We have concentrated on design time context scoping in the current context, i.e. in the situations. The physical context in our context awareness concept correlates with a core ontology in the extended IOP [14]. The core ontology models environmental context that is common and relevant for all kinds of smart space applications; personal, indoor, city (outdoor). The core ontology models the external state of the smart space application. The digital context of environment is new context area and it relates with the internal state of the smart space application. In the extended IOP the digital context of environment approximates with a domain specific ontology. The domain means a type of the smart space in question: personal, indoor or city. The situation context in our context awareness concept corresponds with the application view of domain specific ontology in the extended IOP.

Figure 2 presents the layers of our context awareness concept on the left hand side and the counterparts of the extended IOP are on the right hand side. The context agents are extracted from the Smart City scenario presented above. A Trigger agent is an example of the context based adaptation. The trigger activates the smart car to propose alternative road when the emergency has happened and Susanna is starting her daily trip via the jammed highway. The black arrow, 'New route suggested', in the Figure 2 illustrates the activation of Trigger. Agents in the second (middle) level are an Integrator and a Location estimator and they are variations from the context reasoning agent. They utilize the information provided by first level agents that are interested in a certain context, e.g. a camera view or radio coverage. The outputs, illustrated with black arrows in the Figure 2, form the relevant view of the context information to the application managing emergencies. This view is a scope of an application. The scope can be defined at design time or run time.

Based on this validation we see that our context specific agents for monitoring, reasoning and adaptation fulfilled the requirement to achieve the context-awareness in the smart space.

V. CONCLUSION

In this paper, we introduced 1) the novel context ontology, 2) the concept of context-awareness that takes care of context monitoring, context reasoning and context based adaptation, and 3) the Smart City scenario for validating the proposed concept within the extended IOP, i.e. enhanced Smart-M3 architecture. Our context ontology highlights the derived context through the three context dimensions and levels. We used our novel context ontology successfully for designing the presented Smart City scenario. The validation showed that by using the context aware concept the context-awareness is achieved in the smart space.

In the future, we will concentrate on implementing the abovementioned smart city scenario to proof our context awareness concept. We will proceed our work with the application view of the domain specific ontology to give evidence that it is reachable by a generic agent or a primitive used by the smart (space) applications. We will also work towards 1) the separation of concerns between the context model and service architecture model and 2) standardized context ontology to enhance the different kinds of systems, devices, and sensors to be interoperable in the smart spaces.

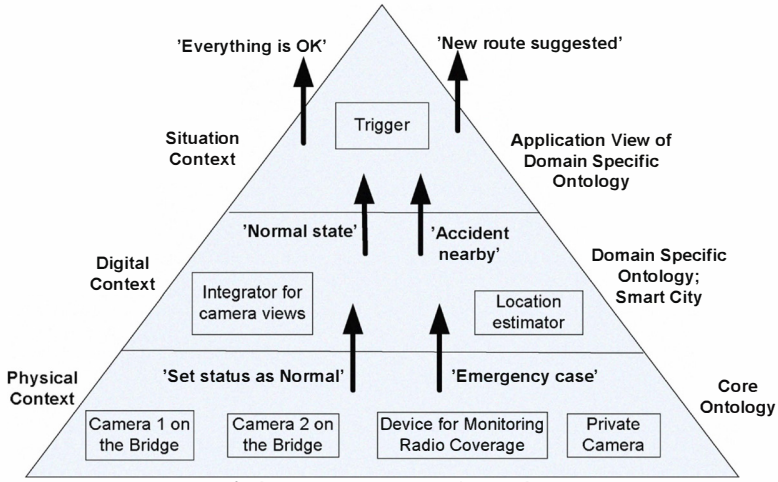

Figure 2. Context agents in the Smart City scenario

ACKNOWLEDGMENT

This work has been carried out in SOFIA/Artemis JU SP3/10017 and the Smash project. SOFIA is funded by Tekes – the Finnish Funding Agency for Technology and Innovation, VTT and European Commission. Smash is funded by VTT.

REFERENCES

[1] S. Hadim, N. Mohamed. "Middleware for wireless sensor networks: A survey. Communication System Software and Middleware", Proc. 1st International Conference on Comsware, 2006. pp. 1-7.

[2] V. Könönen. Multiagent reinforcement learning in Markov games: asymmetric and symmetric approaches. Doctoral thesis, Helsinki University of Technology, Dissertations in Computer and Information Science, Report D8, 2004, Espoo, Finland.

[3] H. Chen, T. Finin, A. Joshi. The SOUPA Ontology for Pervasive Computing. Whitestein Series in Software Agent Technologies, Springer. 2005.

[4] A, Soylu, P. De Causmaecker1, P. Desmet: Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering, JOURNAL OF SOFTWARE, VOL. 4, NO. 9, NOVEMBER 2009, pp.992 – 1013.

[5] Smart-M3, Feb. 2010: http://sourceforge.net/projects/smart-m3/

[6] A. K. Dey, G. D. Abowd, "Towards a better understanding of context and context-awareness", Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, 1999.

[7] C. Bettini, , O. Brdiczka, K. Henricksen, J. Indulska, D. Niclas, A. Ranganathan, D. Riboni. "A survey of context modelling and reasoning techniques". Pervasive Mobile Computing (2009), Doi: 10.1016/j.pmcj.2009.06.002.

[8] D. Preuveneers, Y. Berbers. "Internet of Things: A Context-Awareness Perspective". The Internet of Things: From RFID to the Next Generation Pervasive Networked Systems. L. Yan et al. (eds.) CRC Press. pp. 287-307.

[9] G. Kapitsaki, G. Prezerakos, N. Tselikas, I. Venieris. "Context-aware service engineering: A survey". The Journal of Systems and Software. 82 (2009), pp. 1885-1297.

[10] E. Tanter, J. Noye. A versatile kernel for multi-language AOP. Proc. of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, LNCS, Springer Verlag, pp 173-188.

[11] A. Achillelos, K. Yang, N. Georgalas. Context modelling and a context-aware framework for pervasive service creation: A model-driven approach". Pervasive and Mobile Computing (2009). Doi: 10.1016/j.pmcj.2009.07.014

[12] A. Toninelli, S. Pantsar-Syväniemi, P. Bellavista, E. Ovaska. Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability. Proc. of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing, SESSION: Short papers, Article No: 5, ACM, IFIP, USENIX (2009). http://doi.acm.org/10.1145/1657127.1657134

[13] A. K. Dey, A., Newberger. Support for Context-Aware Intelligibility and Control. CHI 2009, Programming Tools and Architectures, 7th April 2009, Boston, MA, USA.

[14] SOFIA project deliverable D5.21: Interoperable Service Architecture, 2010.

PUBLICATION IX

Case Study: Context-aware supervision of a smart maintenance process

In: 2011 IEEE/IPSJ International Symposium on Applications and the Internet, 18–21 July 2011.
Pp. 309–314.

Case Study: Context-aware Supervision of a Smart Maintenance Process

Susanna Pantsar-Syväniemi, Eila Ovaska
VTT Technical Research Centre of Finland
Oulu, Finland
{susanna.pantsar-syvaniemi, eila.ovaska}@vtt.fi

Susanna Ferrari, Tullio Salmon Cinotti, Guido Zamagni
ARCES, University of Bologna
Bologna, Italy
susanna.ferrari2@studio.unibo.it, {tsalmon, gzamagni}@arces.unibo.it

Luca Roffia, Sandra Mattarozzi, Valerio Nannini
CCC
Bologna, Italy
lroffia@arces.unibo.it
{s.mattarozzi, v.nannini}@bo.icie.it

***Abstract*—The "smartness" of applications is heavily based on their ability to use context information for behaving in a way that perfectly matches the situation in which the application is being used and running. However, achieving situation-based behavior is not straightforward; the situation has to be identified; the functionality that provides the best fit (not always the optimal one) has to be selected from a set of options; and the smart environment has to be configured so that the primary goal can be achieved. This paper introduces an advanced solution for context monitoring and context sharing among software agents; a graphical user interface for defining the scope of context for the application(s) in hand; the matchmaking of ontologies for defining the reasoning rules to be used; and a rule graph as a means of configuring an application's behavior in a way that it is aligned with the goal of the smart environment. The concept has been implemented as part of the maintenance process of a smart building. It is still a challenge to achieve a context that can dynamically update and, above all, expand itself.**

Keywords—adaptation; run-time; context-awareness; micro-architecture; smart environment

I. INTRODUCTION

An environment becomes "smart" when its physical and logical features are available in a digital format for end-user services. In such an environment, information interoperability is a primary requirement. The current trend tends to solve this interoperability requirement with a shared information space, where knowledge about the environment is kept up-to-date and shared with the subscribed and authorized users of the space. Even events activated by the end-user services or applications are shared. Thus, the exchange of information and events through the smart space is beneficial to smart space users only if the provider and consumer of the information or events have a mutual understanding of their meanings. That understanding relies on a shared knowledge, represented as an ontology or set of ontologies.

It is still challenging to design and implement dynamically evolvable and interoperable software for pervasive and smart environments. The ongoing SOFIA project [1] aims to provide an open innovation platform (OIP) including middleware, methods and tools to create smart environments. This paper presents work done within the SOFIA project, where we have utilized the Smart-M3 [2] solution for smart space implementation. The Smart-M3 provides a backbone approach to creating a smart environment. The key enabler is called an SIB (Semantic Information Broker), which is based upon the idea of making statements in the form of subject-predicate-object expressions. These expressions are known as triples in the Resource Description Framework (RDF). The RDF [3] is a directed, labeled graph data format for representing information on the Web.

In this work we report on the utilization of the context-awareness micro-architecture [4] to design and implement a Context Monitor agent for supervising the phases of a building maintenance process. The Context Monitor enhances the already envisaged smart building maintenance process [5, 6]. Furthermore, we introduce a new way for bounding the context information relevant for the context monitoring agent in question. The context is selected with a GUI (Graphical User Interface), called Context Selector. The Context Selector utilizes the context ontology and the application ontology when it creates the selection boxes for a user to select the relevant context to be monitored. Thus, the Context Selector is a general graphical configuring module for selecting the context by which the maintenance person can define the limits of information usage for smart space applications. Any piece of data, at a given time, can be a context for a given smart space application [7]. Hence, the Context Monitor is a novel solution because it is context- (and Smart-M3) based and it is reconfigurable.

The structure of the paper is as follows. Section 2 presents the background. Section 3 introduces our case, where the Context Monitor supervises the smart maintenance of a building. Thereafter, Section 4 goes through the implementation of the Context Selector and Context Monitor and Section 5 discusses the case study. The Conclusion and future work close the paper.

II. BACKGROUND

Ontological models have well-known advantages regarding support for interoperability and heterogeneity [8]. Furthermore, they 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 to high-level context abstractions. Adaptations in context-aware applications are caused by

978-0-7695-4423-6/11 $26.00 © 2011 IEEE
DOI 10.1109/SAINT.2011.59

changes in situations, i.e. a change in a context value triggers adaptation if the context update changes the situation. [8]

Situation-aware applications have benefited from the usage of ontologies [9]. As above-mentioned, we will enhance the existing smart building maintenance approach [5] with the context based agent, Context Monitor, which is able to supervise the phases of the building maintenance. The Context Monitor will be an instance of the context monitoring agent that is one of the dynamic agents of the context-awareness micro-architecture [4]. The context-awareness micro-architecture consists of three types of agents: the context-monitoring, the context-reasoning and the context-based adaptation agents. Fig. 1 shows, using numbered connections, the execution order of the context-awareness agents. Firstly, the context monitoring agent provides information to the semantic database (SIB) to be used by the context reasoning agent. Lastly, the context-based adaptation is notified by the information upgraded by the context-reasoning agent.

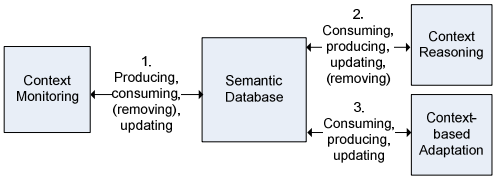

Figure 1. The logical structure of the context-awareness micro-architecture [4]

The context-awareness micro-architecture also has the context ontology which has three main concepts that define the context of the smart environment: the physical-environment context, the digital-environment context and the user context [10]. Following this, we split the user context into two parts according to the interest of the information user; the situational context which is the interest of the application and the user context which merely defines the user intents and preferences [4].

The main concepts of the context ontology are organized according to three levels of abstraction (i.e. context levels) that belong to the corresponding context levels: the physical context of the environment, the digital context of the environment, and the situation context. The physical context contains raw data about the physical environment. The digital context information is a fusion of physical raw data and other state information of the software services combined with some specific reasoning algorithms or rules. The situational context is a set of information derived from physical and digital context information for the application purpose. The user context is (as noted before) relates to the person's interests, intents and activities, not the information of the space itself. The context-awareness agents are illustrated in the corresponding interpretation and abstraction levels in Fig.2.

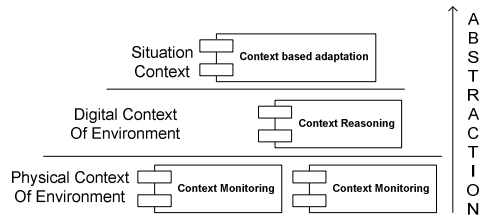

Figure 2. Context-awareness agents on the corresponding context levels [10]

The next section will introduce our solution for scoping the context based on the context ontology and creating a context (i.e. a graph) by mapping the context ontology concepts with the maintenance concepts. Then, this context is used for context-based monitoring and simple reasoning based on the monitoring and behavioral rules.

III. CASE STUDY: CONTEXT-AWARE SUPERVISION OF SMART MAINTENANCE

The goal for the case is to build a smart, context-based agent for supervising the maintenance scenario of a building. The maintenance scenario starts when, e.g., a sensor detects water on the floor. Each fault has its own supervisor. As mentioned above, the smart maintenance system with the maintenance ontology is the legacy software that is enhanced with new features, consisting of the Context Selector and Context Monitor. In order to perform context-aware supervision, we aligned the context and maintenance ontologies to get the new feature to be interoperable with the legacy (smart maintenance) system. After alignment, we had a shared context, i.e. graph, to supervise the progress of the maintenance scenario. Lastly, the progress of the supervision workflow was visualized. A promising tool for that purpose is the Interactive Quality Visualization (IQVis) tool [11] because it supports run-time visualization. The IQVis tool is already used for illustrating how the run-time properties of a smart space change over time [12].

A. Relation to the context ontology and Context Selector

The context to be monitored is defined via a graphical user interface (GUI) of the Context Selector as shown in Fig. 3. The Context Selector reads context concepts from the context ontology file that is in the OWL (Web Ontology Language) [13] format. Then, the Context Selector maps the main context concepts to the corresponding maintenance concepts. The mapping of the concepts is illustrated in Table I.

TABLE I. ALIGNMENT OF CONCEPTS

Maintenance concepts	Context concepts with relations
Person; tenant and maintenance operator	User Context – has Person
Fault	Physical Environment Context – has Event
Corrective Intervention	Physical Environment Context – has Event
Building Room	Digital Environment Context - has Feature

The starting and stopping of an intervention, as well the corrective intervention itself are information from "the digital context of environment" level. The fault is an indication relating to the physical context of the environment. The fault in the building is mostly noticed by the sensors, but it can also be notified by users. The room belongs to "the digital context of environment" level.

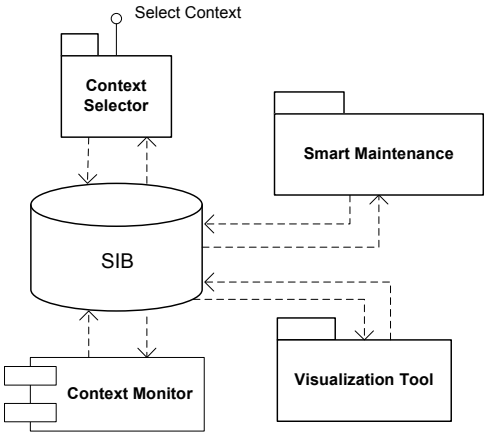

Figure 3. The context of context-aware supervision

The Context Selector formulates the context based on the selections made by a user who knows what he/she wants to monitor. The context is then saved to the SIB. Then, the Context Monitor continues its work with the context.

B. Context Monitoring (and reasoning)

The Context Monitor subscribes to all information showed by the context. It receives notifications from each context change stored to the SIB. The Context Monitor follows the sequence of the maintenance scenario according to the requirements defined for it. The maintenance scenario has many phases and each of them is triggered by a change in the context. For this case study, we chose the following phases as they are the main ones from the viewpoint of the supervision:

- Phase 0: A fault has been detected at time T_0. The Context Monitor starts to wait for the notification "Intervention scheduled". That notification should occur within a specified time, ΔT_0. The notification "Stop intervention" is expected within a pre-defined time, ΔT_{max}. An alert should be generated if that notification does not occur within ΔT_{secure}.
- Phase 1: The maintenance company has set a scheduled time of the intervention, T_x (at time T_1). The maintenance operator should accept to do the intervention within a pre-defined time, ΔT_1.
- Phase 2: One of the maintenance operators has accepted to carry out the intervention.
- Phase 3: The maintenance operator has started the intervention on-site.
- Phase 4: The maintenance operator has successfully completed the intervention and the supervision process is terminated.

Based on knowledge about the ongoing phase and the notifications received from the SIB, the Context Monitor updates the progress of the supervision to the SIB. The Context Monitor inserts "phase passed" if the scenario is going ahead as defined. If the progress is delayed, then the Context Monitor updates the SIB with an alert to be notified to the maintenance company, that in turn has its own tool for visualizing the progress of the maintenance operation and the alerts.

The Context Monitor operates according to its requirements, where the requirements express the behavior in each phase. The behavioral rules depend on time and on the current phase and they are in the following form:

if context change detected **then {**
if expected context-change has occurred, then
{Start or stop timers @ T
Update SIB with phase passed}
Else update SIB with an alert
}
if timer has elapsed **then** update SIB with an alert

As shown in Table II, for each phase the timers that need to be activated or deactivated, are specified, e.g. in P_0 $Timer_1$, $Timer_2$, $Timer_3$ are activated while in P_4 $Timer_2$ and $Timer_3$ are deactivated. The time used for each timer is based on the time at which a phase is detected, e.g. T_0 for phase P_0, and with a delay, e.g. ΔT_{max} related to a specific fault. Some of the faults have to be closed within 24 h, while other faults can have more relaxed or restricted constraints. If a timer expires, or an unexpected context change occurs, the Context Monitor updates the SIB with an alert, e.g. "Start of the intervention delayed" or "Unexpected event".

TABLE II. BEHAVIORAL RULES FOR CONTEXT MONITORING

Rules on phase detection	
P_0	Start timer TR_1 @ $T_0 + \Delta T_0$ Start timer TR_2 @ $T_0 + \Delta T_{max}$ Start timer TR_3 @ $T_0 + \Delta T_{secure}$ Update to the SIB: "Phase passed, Fault detected" Wait for the notification: "Intervention scheduled"
P_1	Stop timer TR_1 Start timer TR_4 @ $T_1 + \Delta T_1$ Start timer TR_5 @ $T_x + \Delta T_x$ Update to the SIB: "Phase passed, Intervention scheduled" Wait for the notification: "Intervention accepted"
P_2	Stop timer TR_4 Update to the SIB: "Phase passed, Intervention accepted" Wait for the notification: "Start intervention"
P_3	Stop timer TR_5 Update to the SIB: "Phase passed, Start intervention" Wait for the notification: "Stop intervention"
P_4	Stop timer TR_2 Stop timer TR_3 Update to the SIB: "Phase passed, Stop of intervention"
Rules on timer expiration	
TR_1	*Alert:* "Scheduling of intervention delayed"
TR_2	*Alert:* "End of intervention delayed"
TR_3	*Alert:* "Warning: close to the available maximum time"
TR_4	*Alert:* "Intervention acceptance delayed"
TR_5	*Alert:* "Start of intervention delayed"

The ultimate aim is to be able to provide the rules via the Context Selector to the SIB, from which they are used as configuration parameters by the Context Monitor or by the separate context-reasoning agent. As a proof of the concept related to the context-awareness architectures and its Context Monitor, the rules are hard coded in this case study.

IV. IMPLEMENTATION

The Context Selector uses the context ontology as a starting point because the context ontology is a general one and can be directly aligned with the application-specific ontology if they share the same main concepts or classes. Fig. 4 shows the view of the Context Selector when the end user starts to select the context for monitoring.

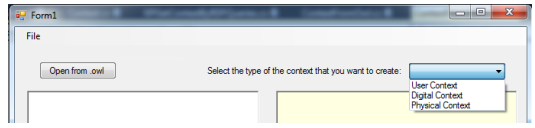

Figure 4. View to select the main context from the context ontology

In this case the maintenance ontology does not share the same main (or core) concepts with the context ontology so we aligned the smart maintenance ontology with the context ontology by hard coding the associations. The hard coding is done by the mapping table as shown in Fig. 5. The KP (knowledge processor) is a term used in the Smart-M3 platform for the piece of software (or agent) that produces or consumes the information from the SIB. The KP can also both consume and produce information.

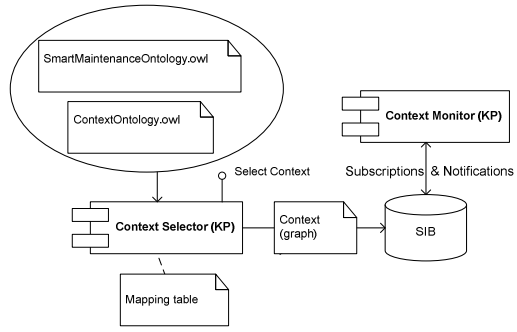

Figure 5. The KPs and SIB in the implementation

For the mapping we used the following definitions: a context attribute and a context argument. Based on these definitions we formulated the context as is exemplified in Fig. 6. Thus, the context arguments and attributes form the context to be monitored.

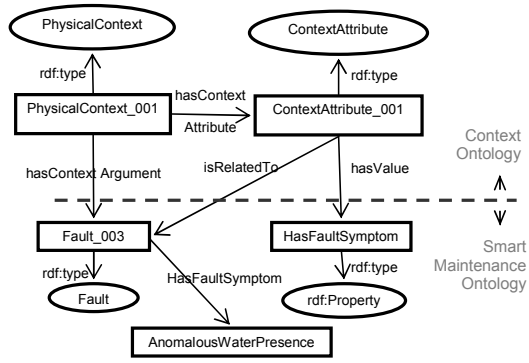

Figure 6. Example of instance-based ontology mapping

The context as a graph is saved to the SIB to be used by the Context Monitor that is the instance of the context monitoring agent. The Context Monitor subscribes to all the selected instances of the context arguments and their attributes as exemplified in Fig. 6. We were able to make selections based on the instances because the Smart Maintenance Ontology was mirroring the content from the SIB. The Context Monitor receives notifications from the SIB according to the subscriptions it has made. The subscriptions and notifications are mechanisms available in the Smart-M3 platform [2].

The context is semi-automatic and is created from information instances, as we did in this case. By semi-automatic context we mean that the context is not automatically updated when a new kind of a sensor is added to the maintained building if the sensor type is not one of the instances available in the smart-maintenance ontology.

The Context Selector and the Context Monitor were written in C# in the framework .NET 4.0 and they used C# libraries to access the SIB. All these components, including the SIB, were running on Ubuntu 9.04. The Context Selector used dotNetRDF library [14] to read the owl-files.

V. DISCUSSIONS

This case study convinced us of the usefulness of context-awareness micro-architecture. The Context Monitor is a reusable and reconfigurable software agent that is updated at run-time by application-specific parameters represented as a context graph. The Context Monitor and the Context Selector were implemented in this case with the Smart-M3 platform-based SIB (semantic database) but they can be implemented with other RDF-based platforms as well.

The Context Monitor, instantiated from the context-aware micro-architecture, has many advantageous characteristics:

- It is a reusable solution, i.e. it can be applied to different application domains and platforms;
- It follows a separation-of-concerns principle; a solution defined on the model level, i.e. an application model, a context model and a platform model;
- It is independent from the implementation languages;
- It is a scalable solution i.e. it can be instantiated several times for different monitoring purposes within one application if needed
- Its behavior is configurable using the rules.

The mechanism of the Context Selector to scope the situational context (the view of the interesting context information) is a successful solution. The novelty of the Context Selector comes from the ability to work with the ontology files and still to be able to formulate the configuration of the context. The challenging task was to map the context ontology with the smart maintenance ontology because they did not share the same concepts. This is because the maintenance ontology was a legacy one and developed from the building domain viewpoint at a time when there was no suitable context ontology available. The mapping between these two different ontologies is more straightforward when they use the same context ontology.

The context can be also created based on classes, not only from instances (in Fig. 6, for example, the context is based on a fault instance). The selection between classes or instances is a consequence of the used application ontology and the decision for the content of that ontology can be made on a need basis. With class-based selection, the context is automatically updated each time a new instance is created from the selected class, but then the context cannot be as full of properties as it is with instance-based context. This is due to the fact that the Context Monitor is not able to make subscriptions to the properties of the class because the properties are not transformed in the context (graph).

The rules in the Context Monitor were hard coded this time but we are working towards a reusable agent for context reasoning that can be dynamically updated via the SIB (semantic interface).

VI. CONCLUSIONS AND FUTURE WORK

Due to the dynamism of smart environments, the software agents' behavior needs to be flexible and adaptable in a situation-based manner. This means that the situation is to be identified and analyzed, and the actions to be taken are to be reasoned and finally executed according to the common 'requirements and constraints' set for the smart environment. Commonalities can be described as ontological models. However, the commonality also changes, i.e. the commonality of today is not necessarily the commonality of tomorrow. Thus, ontology matchmaking is required.

In this paper, a novel concept on context-awareness architecture was demonstrated in a smart maintenance scenario by implementing a context-aware supervision feature. A generic context-monitoring agent, Context Monitor, was used for collecting the context information stored for the shared use of agents into an SIB. An independent application (Context Selector), with a graphical user interface, was implemented for defining the scope of the context for the application in hand and mapping the selected context to the application-specific data collected to the same shared repository. The context graph, defined as a result of the mapping activity, was used for configuring the context-monitoring agent, Context Monitor.

To achieve a context that can dynamically update and, above all, expand itself is a challenging task. With instance-based ontology that kind of context is not possible, however, it is possible with class-based ontology, but the context is limited to the amount of the properties it has. We are working on class-based ontology to achieve a dynamic context that updates and expands (evolves) itself at run-time. The context is a basis for context monitoring, context reasoning and context-based adaptation. We are also working to get dynamic rules beside the context to the SIB. In addition, we will also consider social aspects to be taken into account in the context ontology.

ACKNOWLEDGMENT

This work has been carried out in the SOFIA/Artemis JU SP3/10017 project. SOFIA is funded by Tekes – the Finnish Funding Agency for Technology and Innovation, VTT, MIUR – Italian Ministry for Education and Research, and the European Commission.

REFERENCES

[1] SOFIA project, http://www.sofia-project.eu/

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

[3] Resource description framework, http://www.w3.org/RDF/

[4] S. Pantsar-Syväniemi, J. Kuusijärvi, and E. Ovaska, "Context-awareness Micro-Architecture for Smart Spaces", Proc. 6th Int. Conference on Grid and Pervasive Computing (GPC 2011), May 11-13, 2011, Oulu, Finland, in press.

[5] D. Manzaroli, L. Roffia, T. Salmon Cinotti, E. Ovaska, P. Azzoni, V, Nannini and S. Mattarozzi, "Smart-M3 and OSGi: The Interoperability Platform", Proc. IEEE Symp. 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010), IEEE Press, 2010, pp. 1053-1058, doi: 10.1109/ISCC.2010.5546633.

[6] A. D'Elia, L. Roffia, G. Zamagni, F. Vergari, A. Toninelli, and P. Bellavista, "Smart applications for the maintenance of large buildings: How to achieve ontology-based interoperability at the information level", Proc. IEEE Symp. 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010), IEEE Press, 2010, pp. 1077-1082, doi: 10.1109/ISCC.2010.5546633.

[7] A. Toninelli, S. Pantsar-Syväniemi, P. Bellavista, and E. Ovaska, "Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability", Proc. Int. Workshop Middleware for Pervasive Mobile and Embedded Computing (M-MPAC 2010), session: short papers, Article No: 5. ACM, IFIP, USENIX, 2009, doi: 10.1145/1657127.1657134.

[8] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," Pervasive and Mobile Computing, vol. 6, Apr. 2010, pp. 161-180, doi:10.1016/j.pmcj.2009.06.002.

[9] N. Baumgartner, W. Retschitzegger, and W. Schwinger, "A Software Architecture for Ontology Driven Situation Awareness", Proc. ACM Symp. Applied Computing (SAC'08), ACM, New York, 2008, pp. 2326-2330 , doi: 10.1145/1363686.1364237.

[10] S. Pantsar-Syväniemi, K. Simula, and E. Ovaska, "Context-Awareness in Smart Spaces". Proc. IEEE Symp. 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010), IEEE Press, 2010, pp. 1023-1028, doi: 10.1109/ISCC.2010.5546630.

[11] J. Kuusijärvi, "Interactive Visualization of Quality Variability at Run-time," University of Oulu, Department of Electrical and Information Engineering, Master's Thesis, 86 p, 2010.

[12] J. Kuusijärvi, A.Evesti, and E. Ovaska."Visualizing Structure and Quality Properties of Smart Spaces". Proc. IEEE Symp. 1st Int. Workshop Semantic Interoperability for Smart Spaces (SISS 2010), IEEE Press, 2010, pp. 1023-1028, doi: 10.1109/ISCC.2010.5546630.

[13] Web ontology language, http://www.w3.org/2004/OWL/

[14] An Open Source C#.Net Library for RDF, http://www.dotnetrdf.org/

Series title and number
VTT Science 37

Title	**Reusable, semantic, and context-aware micro-architecture** **Approach to managing interoperability and dynamics in smart spaces**
Author(s)	Susanna Pantsar-Syväniemi
Abstract	The amount of shared information has increased a great deal in ubiquitous systems, where the previously isolated devices and appliances have become part of the system and are producing or consuming the information. The ubiquitous system, or the smart environment, lacks an approach that supports scalability and enables semantic interoperability. It is challenging to provide a dynamic behavior at the run time without human intervention. A number of dedicated solutions have been developed for the ubiquitous environment because of its complexity. The dedicated solutions are usually non reusable. An approach is needed that i) is reusable as such or partly, ii) provides the semantic interoperability, iii) enables dynamic and behavioral interoperability between the receiver and sender of the information at run time, and iv) is scalable by being modular, and decoupled. This thesis proposes a novel approach to managing interoperability and dynamics in smart spaces. The approach includes a Context-Aware Micro-Architecture (CAMA), and a Context Ontology for Smart Spaces (CO4SS). This approach is independent of implementation languages and communication techniques. CAMA, as an architectural pattern, is usable without its semantic support, CO4SS. In the literature, it is the first approach that fulfills the requirements that are set for a context data distribution system. The power in CAMA relies on the usage of the standard and web-based techniques, in the separation-of-concerns principle, and in the enhanced control loop, MAPE-K. The latter has four parts, Monitor; Analyze; Plan; Execute that share Knowledge. CAMA is highly dynamic, which is due to the run-time updatable rules. The creation of the rules is laborious, as they are written into text boxes of Message Sequence Charts. This will be improved when new tools are developed for the rule creation. Additional research is needed to validate the scalability of the approach with a “Big data”. CO4SS can be widened with the domain-specific and quality ontologies. It supports the evolution management of the smart space: all smart spaces and their applications ‘understand’ the common language that is defined by it. CO4SS has the potential to be a de facto ontology for the context-aware, i.e., intelligent applications.
ISBN, ISSN	ISBN 978-951-38-8009-5 (Soft back ed.) ISBN 978-951-38-8010-1 (URL: http://www.vtt.fi/publications/index.jsp) ISSN-L 2242-119X ISSN 2242-119X (Print) ISSN 2242-1203 (Online)
Date	August 2013
Language	English, Finnish abstract
Pages	72 p. + app 122 p.
Key words	Ontology, software architecture, embedded, ubiquitous system, design pattern
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 37

Nimeke	**Uudelleenkäytettävä, semanttinen ja kontekstitietoinen pienoisarkkitehtuuri** **Menetelmä yhteentoimivuuden ja dynamiikan hallintaan älykkäissä tiloissa**
Tekijä(t)	Susanna Pantsar-Syväniemi
Tiivistelmä	Kaikkialla läsnä oleva, ns. Ubi-järjestelmä, sisältää paljon yhteistä tietoa, jonka määrä kasvaa, kun ennen erillään toimineet laitteet tulevat osaksi järjestelmää. Ubi-järjestelmä, tai älykäs tila, tarvitsee uusia menetelmiä, jotka tukevat järjestelmän skaalautuvuutta sekä mahdollistavat semanttisen yhteentoimivuuden eri laitteiden ja applikaatioiden välillä. Tällainen järjestelmä on dynaaminen, ja on haasteellista saada sitä tukevaa automaattista toimintaa toimimaan reaaliajassa. Ubi-järjestelmä on myös monimutkainen, ja siksi olemassa olevat järjestelmät ovat olleet hyvin erikoistuneita, eivät uudelleenkäytettäviä. Tarvitaan siis ratkaisumalli, joka i) on uudelleenkäytettävä sellaisenaan tai osittain, ii) tarjoaa semanttisen yhteentoimivuuden, iii) mahdollistaa dynaamisen ja yhteentoimivan toiminnan reaaliajassa tiedon lähettäjän ja vastaanottajan välillä ja iv) on skaalautuva. Väitöstyössä on kehitetty uusi menetelmä yhteentoimivuuden ja dynamiikan hallintaan älykkäissä tiloissa. Menetelmässä on kontekstitietoinen mikroarkkitehtuuri (CAMA) sekä kontekstiontologia (CO4SS). Ne ovat riippumattomia toteutuskielistä ja kommunikointiteknologioista. CAMA on arkkitehtuurimalli, ja se on käytettävissä ilman semanttista tukeaan eli kontekstiontologiaa, CO4SS. Menetelmä on kirjallisuudessa ensimmäinen, joka täyttää kontekstitiedon jakelujärjestelmän vaatimukset. Mikroarkkitehtuurin vahvuus on siinä, että se käyttää sekä standardoituja että web-pohjaisia tekniikoita. Arkkitehtuuri pitää kontekstinhallinnan erillään muusta ohjelmistosta ja hyödyntää parannettua MAPE-K mallia, jonka neljä osaa – monitorointi (M), analysointi (A), suunnittelu (P) ja toteutus (E) – hyödyntävät tietämystä (K). Reaaliaikaisesti päivitettävät säännöt tekevät kehitetystä mikroarkkitehtuurista hyvin dynaamisen. Sääntöjen luonti on työlästä, koska säännöt kuvataan tekstilaatikkoina toimintajärjestyskuviin. Tämä paranee tulevaisuudessa, kunhan uusia työkaluja sääntöjen kuvaamiseen saadaan kehitettyä. Menetelmän skaalautuvuutta suurien datamäärien kanssa on vielä tarpeen tutkia lisää. Kontekstiontologia on laajennettavissa niin sovellusalue- kuin laatuontologioilla. Se tukee älytilojen evoluutiota tarjoamalla yleisen kielen, jota kaikki älytilat ja niihin liittyvät ohjelmistot ymmärtävät. Ontologialla on edellytyksiä kehittyä ns. de to -kontekstiontologiaksi, kun luodaan kontekstitietoisia eli älykkäitä ohjelmistoja.
ISBN, ISSN	ISBN 978-951-38-8009-5 (nid.) ISBN 978-951-38-8010-1 (URL: http://www.vtt.fi/publications/index.jsp) ISSN-L 2242-119X ISSN 2242-119X (painettu) ISSN 2242-1203 (verkkojulkaisu)
Julkaisuaika	Elokuu 2013
Kieli	Englanti, suomen kielinen tiivistelmä
Sivumäärä	72 s. + liitt. 122 s.
Avainsanat	Ontology, software architecture, embedded, ubiquitous system, design pattern
Julkaisija	VTT PL 1000, 02044 VTT, Puh. 020 722 111

Reusable, semantic, and context-aware micro-architecture

Approach to managing interoperability and dynamics in smart spaces

This thesis proposes a novel approach to managing interoperability and dynamics in smart spaces. The approach i) is reusable as such or partly, ii) provides the semantic interoperability, iii) enables dynamic and behavioral interoperability between the receiver and sender of the information at run time, and iv) is scalable by being modular, and decoupled.

The developed approach includes a Context-Aware Micro-Architecture (CAMA), and a Context Ontology for Smart Spaces (CO4SS). This approach is independent of implementation languages and communication techniques. CAMA, as an architectural pattern, is usable without its semantic support, CO4SS.

The power in CAMA relies on the usage of the standard and web-based techniques, in the separation-of-concerns principle, and in the enhanced control loop, MAPE-K. The latter has four parts, Monitor; Analyze; Plan; Execute that share Knowledge. CAMA is highly dynamic, which is due to the run-time updatable rules. The creation of the rules is laborious, as they are written into text boxes of Message Sequence Charts. This will be improved when new tools are developed for the rule creation. Additional research is needed to validate the scalability of the approach with a “Big data”. CO4SS can be widened with the domain-specific and quality ontologies. It supports the evolution management of the smart space: all smart spaces and their applications ‘understand’ the common language that is defined by it. CO4SS has the potential to be a de facto ontology for the context-aware, i.e., intelligent applications.

ISBN 978-951-38-8009-5 (Soft back ed.)
ISBN 978-951-38-8010-1 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN-L 2242-119X
ISSN 2242-119X (Print)
ISSN 2242-1203 (Online)