OULU 2015
C 545

ACTA UNIVERSITATIS OULUENSIS

Ekaterina Gilman

EXPLORING THE USE OF RULE-BASED REASONING IN UBIQUITOUS COMPUTING APPLICATIONS

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACTA UNIVERSITATIS OULUENSIS
C Technica 545

EKATERINA GILMAN

EXPLORING THE USE OF RULE-BASED REASONING IN UBIQUITOUS COMPUTING APPLICATIONS

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in the OP auditorium (L10), Linnanmaa, on 30 October 2015, at 12 noon

UNIVERSITY OF OULU, OULU 2015

Acta Univ. Oul. C 545, 2015

Supervised by
Professor Jukka Riekki

Reviewed by
Professor Claudio Bettini
Professor Tommi Mikkonen

Opponent
Professor Arkady Zaslavsky

ISBN 978-952-62-0957-9 (Paperback)
ISBN 978-952-62-0958-6 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2015

Gilman, Ekaterina, Exploring the use of rule-based reasoning in ubiquitous computing applications.
University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Computer Science and Engineering
Acta Univ. Oul. C 545, 2015
University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

Ubiquitous computing transforms physical environments into smart spaces, supporting users in an unobtrusive fashion. Such support requires sensing and interpreting the situation of the user, and providing the required functionality utilizing resources available. In other words, context acquisition, context modelling, and context reasoning are required.

This thesis explores rule-based context reasoning from three perspectives: to implement the functionality of ubiquitous applications, to support the creation of ubiquitous applications, and to achieve self-adaptation. First, implementing functionality with reasoning is studied by comparing an application equipped with rule-based reasoning with an application providing similar functionality with hard coded application logic. The scalability of rule-based reasoning is studied with a large-scale student assistant scenario. Reasoning with constrained resources is explored with an application that performs reasoning partially on mobile devices. Finally, distributing a reasoning component that supports smart space interaction is explored with centralized, hybrid, and distributed architectures.

Second, the creation of applications with rule-based reasoning is explored. In the first study, rules support building applications from available services and resources based on the instructions that users give via physical user interfaces. The second study supports developers, by proposing middleware that dynamically selects services and data based on the rules written by application developers.

Third, self-adaptation is explored with a conceptual framework that adds self-introspective monitoring and control to smart space applications. This framework is verified with simulation and theoretical studies, and an application that fuses diverse data to provide fuel-efficient driving recommendations and adapts decision-making based on the driver's progress and feedback.

The thesis' contributions include demonstrative cases on using rule-based reasoning from different perspectives, different scales, and with different architectures. Frameworks, a middleware, simulations, and prototypes provide the concrete contribution of the thesis. Generally, the thesis contributes to understanding how rule-based reasoning can be used in ubiquitous computing. The results presented can be used as guidelines for developers of ubiquitous applications.

Keywords: context reasoning, context-awareness, rule-based reasoning, ubiquitous computing

Gilman, Ekaterina, Sääntöpohjaisen päättelyn hyödyntäminen jokapaikan tietotekniikassa.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta, Tietotekniikan osasto
Acta Univ. Oul. C 545, 2015
Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Jokapaikan tietotekniikka muokkaa fyysisen ympäristömme älykkääksi tilaksi, joka tukee käyttäjää häntä häiritsemättä. Tuki toteutetaan asentamalla ympäristöön käyttäjää ja ympäristöä havainnoivia laitteita, tulkitsemalla kerätyn tiedon perusteella käyttäjän tilanne ja tarjoamalla tilanteeseen sopiva toiminnallisuus käyttäen saatavilla olevia resursseja. Toisin sanoen, älykkään tilan on kyettävä tunnistamaan ja mallintamaan toimintatilanne sekä päättelemään toimintatilanteen perusteella.

Tässä työssä tutkitaan sääntöpohjaista päättelyä toimintatilanteen perusteella sovellusten toiminnallisuuden toteutuksen, kehittämisen tuen sekä mukautuvuuden näkökulmista. Sovellusten toiminnallisuuden toteuttamista päättelemällä tutkitaan vertaamalla sääntöpohjaisen päättelyn avulla toteutettua toiminnallisuutta vastaavaan suoraan sovellukseen ohjelmoituun toiminnallisuuteen. Sääntöpohjaisen päättelyn skaalautuvuutta arvioidaan laajamittaisessa opiskelija-assistenttiskenaariossa. Niukkojen resurssien vaikutusta päättelyyn arvioidaan päättelemällä osittain mobiililaitteessa. Älykkään tilan vuorovaikutusta tukevan päättelykomponentin hajauttamista tutkitaan keskitetyn, hybridi- ja hajautetun arkkitehtuurin avulla.

Sovelluskehityksen tukemiseksi päättelyn säännöt muodostetaan saatavilla olevista palveluista ja resursseista käyttäjän fyysisen käyttöliittymän välityksellä antamien ohjeiden mukaisesti. Toisessa tapauksessa sovelluskehitystä tuetaan väliohjelmistolla, joka valitsee palvelut ja datan dynaamisesti sovelluskehittäjien luomien sääntöjen perusteella.

Mukautuvuutta tutkitaan tilan hallintaan ja itsehavainnointiin liittyvän toiminnallisuuden lisäämiseen pystyvän käsitteellisen kehyksen avulla. Kehyksen toiminta varmennetaan simulointien sekä teoreettisten tarkastelujen avulla. Toteutettu useita datalähteitä yhdistävä sovellus antaa ajoneuvon kuljettajalle polttoaineen kulutuksen vähentämiseen liittyviä suosituksia sekä mukautuu kuljettajan ajotavan kehityksen ja palautteen perusteella.

Työssä on osoitettu sääntöpohjaisen päättelyn toimivuus eri näkökulmista, eri skaalautuvuuden asteilla sekä eri arkkitehtuureissa. Työn konkreettisia tuloksia ovat kehykset, väliohjelmistot, simuloinnit sekä prototyypit. Laajemmassa mittakaavassa työ edesauttaa ymmärtämään sääntöpohjaisen päättelyn soveltamista ja työn tuloksia voidaankin käyttää suosituksina sovelluskehittäjille.

Asiasanat: jokapaikan tietotekniikka, päättely tilannetiedosta, sääntöpohjainen päättely, tilannetietoisuus

Acknowledgements

The research work presented in this thesis was conducted in the Interactive Spaces (iSpaces) research group at the Department of Computer Science and Engineering of the University of Oulu, Finland. Particularly, the research was conducted within DIEM, PSC, Active Learning Spaces, and D2I projects.

First of all, I would like to thank my supervisor, Prof. Jukka Riekki, for his patient support, valuable advice and guidance throughout my doctoral studies.

I would like to acknowledge all colleagues from Interactive Spaces and other research groups with whom I worked during these years. Specifically, I am very grateful to Dr. Oleg Davidyuk, Mr. Xiang Su, Mr. Iván Sánchez, Dr. Susanna Pirttikangas, Dr. Anja Keskinarkaus, Dr. Satu Tamminen, Dr. Jiehan Zhou, Mr. Mikko Pyykkönen, Ms. Marta Cortés, Mr. Timo Saloranta, and Mr. Jussi Mäkipelto. Your contribution has been vital to develop the ideas and prototypes presented in this thesis. I would like to thank all the support staff of the Department of Computer Science and Engineering for making the working life smoothly organized.

I would like to thank the official reviewers, Professor Tommi Mikkonen from Tampere University of Technology and Professor Claudio Bettini from the University of Milan for their valuable comments. I would like to thank Professor Arkady Zaslavsky, Senior Principal Research Scientist at the Commonwealth Scientific and Industrial Research Organisation (CSIRO) for serving as the opponent in the doctoral defence.

This dissertation was partially funded by the Graduate School in Electronics, Telecommunications and Automation (GETA), the Tauno Tönning Foundation, and the Walter Ahlström Foundation. The financial support is greatly appreciated.

I am deeply thankful to my mother, Irina, and my brother, Kirill, and all my family members for their love and support. I thank my friends for making Oulu a second home. I'm thankful to my husband, Alexander, for his love and encouragement during these years.

Oulu, September 2015 Ekaterina Gilman

Abbreviations

AI	*Artificial Intelligence*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
DIEM	*Devices and Interoperability Ecosystem*
DL	*Description Logics*
D2I	*Data to Intelligence*
FPGA	*Field-Programmable Gate Array*
GUI	*Graphical User Interface*
HCI	*Human-Computer Interaction*
IBM	*International Business Machines Corporation*
ICT	*Information and Communications Technology*
IoP	*Internet of People*
IoT	*Internet of Things*
M2M	*Machine to Machine*
NFC	*Near Field Communication*
OWL	*Web Ontology Language*
OWL-S	*Web Ontology Language for Services*
P2P	*Peer-to-Peer*
PSC	*Pervasive Service Computing*
QoS	*Quality of Service*
RDF	*Resource Description Framework*
RFID	*Radio-Frequency Identification*
RIF	*Rule Interchange Format*
SOAP	*Simple Object Access Protocol*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*

List of main terms

Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves [1].

Context-aware system is a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [1].

Context reasoning is deducing new and relevant information for the use of applications and users from the various sources of context data [2].

Reasoning (inferencing) involves logical operations on logical sentences or statements within a model (describing the application domain) in order to draw conclusions and derive other sentences [3].

Rule-based reasoning is a logic-based reasoning technique where rules are used to define the production of new statements and conclusions from initially known statements of the model (describing the application domain) or from the ones derived earlier.

Service is a self-contained unit of functionality that can be accessed with a software interface.

Smart space is a physical environment enriched with ubiquitous applications.

Ubiquitous application is an application which utilizes context information and resources of the environment to support a user.

Ubiquitous service is a service which utilizes context information and resources of the environment to provide its functionality.

User support is provision of functionality and interaction capabilities that match the needs and situation of the user.

Original publications

This thesis is based on seven articles published in peer-reviewed international workshops, conferences, and journals. These articles are listed below and they are referred throughout the text by their Roman numerals (I–VII):

I Gilman E, Sánchez I, Saloranta T & Riekki J (2010) Reasoning for smart space application: comparing three reasoning engines CLIPS, Jess and Win-Prolog. Proc IEEE International Conference on Computer and Information Technology. Bradford, UK: 1340–1345, DOI: 10.1109/CIT.2010.240.

II Gilman E, Su X, Davidyuk O, Zhou J & Riekki J (2011) Perception framework for supporting development of context-aware Web services. International Journal of Pervasive Computing and Communications, 7 (4): 339–364, DOI: 10.1108/17427371111189665.

III Davidyuk O, Gilman E, Sánchez I, Mäkipelto J, Pyykkönen M & Riekki J (2012) iCompose: context-aware physical user interface for application composition. Central European Journal of Computer Science, 1 (4): 442–465, DOI: 10.2478/s13537-011-0031-z.

IV Gilman E & Riekki J (2012) Is there meta-level in smart spaces? Proc IEEE International Conference on Pervasive Computing and Communications Workshop on Managing Ubiquitous Communications and Services, Lugano, Switzerland: 88–93, DOI: 10.1109/PerComW.2012.6197638.

V Gilman E, Davidyuk O, Su X & Riekki J (2013) Towards interactive smart spaces. Journal of Ambient Intelligence and Smart Environments, 5 (1): 5–22, DOI: 10.3233/AIS-120189.

VI Gilman E, Sánchez I, Cortés M & Riekki J (2015) Towards user support in ubiquitous learning systems. IEEE Transactions on Learning Technologies, 8 (1): 55–68, DOI: 10.1109/TLT.2014.2381467.

VII Gilman E, Keskinarkaus A, Tamminen S, Pirttikangas S, Röning J & Riekki J (2015) Personalised assistance for fuel-efficient driving. Journal of Transportation Research Part C: Emerging Technologies, 58 (D): 681–705, DOI: 10.1016/j.trc.2015.02.007.

Contents

1 Introduction

1.1 Background and motivation

The ubiquitous computing paradigm [4] introduces environments able to assist their users in an unobtrusive and proactive way. These environments are called smart environments or smart spaces [5]; the latter term is used in this thesis. Smart spaces are enriched with technology able to recognize the events occurring in the space and to react accordingly. These spaces interact with their users, providing them the right services in the right situations. Smart spaces act as containers for ubiquitous applications, and supply unobtrusive support for the users based on contextual information. In order to support their users, smart spaces should be able to sense and interpret the situation the user is in, identify his or her needs, and provide the required functionality utilizing resources available. In the ubiquitous computing domain, these stages are called context acquisition, context modelling, and context reasoning (Figure 1). Context is the term used to characterize the situation in the smart space. Accordingly, systems which are able to detect and react to context changes are called context-aware [1].

Context acquisition (Figure 1) refers to the process of retrieving the context information by using physical sensors and available services. Context information can also be provided manually by the user, or deduced from available knowledge. Often some data processing is required to transform raw sensor data to context information.

Context information, to distinguish from raw sensor data, can be regarded as a well-structured concept describing a property of an environment or a user [6]. This requires constructing a formal representation of context information, facilitating its storage and operation, that is *context modelling* (Figure 1) [7].

Context reasoning[1] deduces new and relevant information for the use of applications and users from the various sources of context data [2]. Hence, this stage is mainly responsible for interpreting the situation in a smart space, and deciding how to assist its users (Figure 1). Viterbo [8] generalizes the use for context reasoning as follows: 1) to deal with imperfection and uncertainty of context data, 2) to determine the higher-level context, and 3) to trigger adaptation actions.

[1] In this thesis, *context reasoning* and *reasoning on context* are used interchangeably.

Dealing with imperfection and uncertainty is required due to the low reliability of sensors that may cause the perceived context to be erroneous, imprecise, ambiguous or unknown (i.e., no information exists) [9]. For such cases, reasoning may detect the errors and possible conflicts, and validate the retrieved context information.

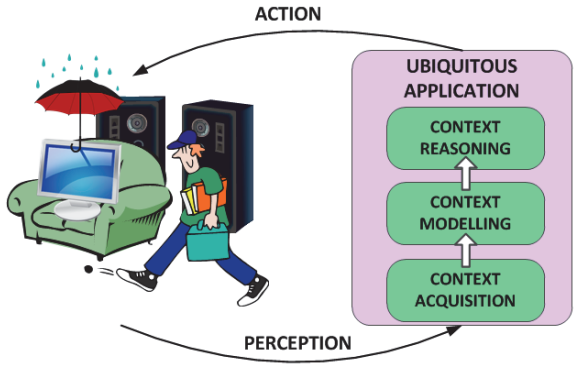

Fig. 1. General ubiquitous application.

Determining higher-level context is necessary to retrieve more meaningful information for applications. Context reasoning provides mechanisms to infer implicit context information from the already stored context; for instance identifying the situation by interpreting or fusing several pieces of context information [6].

Triggering adaptation actions based on situation is responsible for adaptive behaviour of the ubiquitous system. Context reasoning can be used to encode the application logic of a ubiquitous application facilitating such adaptive behaviour. That is, reasoning can be responsible for selecting and performing the appropriate actions for each situation; for instance, for determining whether or not to alert the user about a certain event [10].

There are plenty of techniques which can be used to reason about context in the ubiquitous domain, like rule-based reasoning, neural networks and case-based reasoning [11],[12]. Usually, the choice of the appropriate reasoning technique depends on both the context model and requirements of the concrete application. This thesis concentrates on rule-based context reasoning in ubiquitous applications. This reasoning technique uses rules to infer conclusions from known premises and is based on mathematical logic [13]. A system using a rule-based approach for knowledge representation and reasoning is called a rule-based system.

Even though a large body of knowledge on rule-based systems exists, more research is required on rule-based reasoning in the ubiquitous computing domain. We consider that reference models and recommendations on use of rule-based reasoning would facilitate determining the opportunities, benefits and shortcomings that rule-based reasoning provides for ubiquitous applications. First, research is needed on advantages and drawbacks of rule-based reasoning for ubiquitous applications in general. Even though research exists on analysing the performance of concrete rule engines [14],[15] it is beneficial to analyse the benefits and shortcomings that rule-based reasoning brings to ubiquitous applications in general.

Second, ubiquitous computing introduces technical challenges, like usage of resource-constrained devices and heterogeneous execution environments. Hence, research is required on the scope of rule-based reasoning in ubiquitous computing. For instance, decision-making is expensive in terms of resource utilization, like CPU cycles and memory consumption, hence not all resources of a smart space can perform reasoning tasks. This guides the research for lightweight reasoning engines, like Androjena [16]. The interested reader is referred to Iglesias et al. [17] for a review. Also, large-scale scenarios may have their own challenges for rule-based reasoning.

Third, smart spaces contain heterogeneous resources that interact with each other and with the users. Equipping these resources with context acquisition and decision-making capabilities could facilitate different forms of this interaction. Research is required on whether rule-based reasoning can be useful for this. However, interaction in smart spaces is a very broad research field. For instance, when interaction facilities are embedded into objects of a smart space, the users need to be communicated about such interaction capabilities while preserving the natural outlook of the object as much as possible [18].

Fourth, creating ubiquitous applications is still challenging. Several attempts have been made to support both end users [19],[20] and developers [21],[22]. To support the creation of a holistic view on use of rule-based reasoning in ubiquitous computing, it is necessary to understand how rule-based reasoning can be used to provide the support for creating ubiquitous systems for both users and developers. Ubiquitous computing applications are targeted towards everyday environments and users who do not necessary have specific technical and domain knowledge [23]. These users should be able to use the system, understand how it behaves, and modify it if necessary; otherwise they lose the feeling of being able to control the application [24], [25]. Research is needed on how rule-based reasoning can support

users to create ubiquitous applications by interacting with physical objects in a smart environment. Developers, in their turn, need convenient tools to create, deploy, and maintain ubiquitous applications.

Fifth, smart environments are volatile and fluctuating execution environments for ubiquitous applications [26]. This means that it is difficult to predefine all the available resources, services and their interaction in the design time of the application. Generally, this requires ubiquitous systems to be able to adapt themselves at runtime to meet the changing requirements [27], [28]. To create a holistic view on the use of rule-based reasoning in ubiquitous computing, research is required on how rule-based reasoning can tackle this self-adaptation issue.

There are other important issues to address to build a holistic view on the use of rule-based reasoning in ubiquitous computing. For example, how rule-based reasoning can cope with context ambiguity and inconsistency [29] or support privacy [30], [31]. These all are interesting and difficult research issues which the research community is facing today.

This thesis explores some of the issues regarding rule-based reasoning in ubiquitous computing. We consider different uses of rule-based reasoning to design and develop ubiquitous applications. The presented research is multidisciplinary, and is based on methods and scientific practices of such fields as ubiquitous computing and ambient intelligence, artificial intelligence, software engineering, computer science and human-computer interaction. The scope and objectives of this research are described further in Section 1.2.

1.2 Research questions and scope of thesis

The focus of this thesis is to investigate how rule-based reasoning can be used in ubiquitous computing. The aim is to generate new knowledge which would help to develop useful ubiquitous applications.

We explore the use of rule-based reasoning in ubiquitous computing from three perspectives (see Figure 2). First, it is important to understand what rule-based reasoning brings to smart space applications in general. This means exploring the usage of rule-based reasoning to implement the functionality of ubiquitous applications (Figure 2, bottom). It is necessary to understand what benefits rule-based reasoning provides over conventional hard-coded mechanisms, what are the challenges to using rules for large-scale scenarios and in resource-constrained devices.

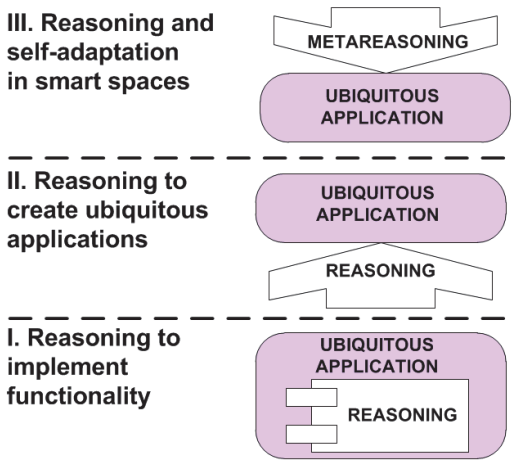

Fig. 2. Perspectives on reasoning for ubiquitous system.

The second aspect is to understand how rule-based reasoning can be used to create ubiquitous applications (Figure 2, middle). This view is important for both developers and users of ubiquitous applications.

The third perspective is related to the ability of smart spaces or applications themselves to monitor their performance and ability to adapt them to perform better (Figure 2, top). Smart spaces are full of services and applications interacting with each other and users, hence to achieve good performance, the overall interaction should be considered. As a set of applications interacting with a user depends on context, it is difficult to tailor the overall interaction beforehand. Hence, it must be considered at runtime. This thesis studies adding an additional rule-based control layer to smart spaces or their applications and services, which monitors and controls the tasks performed by services and applications of smart space in order to improve their performance.

These perspectives lead to the following research questions:

RQ 1. What benefits and constraints does rule-based reasoning bring to the development of ubiquitous applications?

RQ 2. How can rule-based reasoning be used to create ubiquitous applications for smart spaces?

RQ 3. How can rule-based reasoning be used to support the self-adaptation of smart spaces?

In order to answer the first research question, we need to understand the benefits and constraints that rule-based reasoning has when compared with conventional approaches for ubiquitous applications development. Moreover, in order to understand the scope of rule-based reasoning in smart spaces we should explore the challenges of performing reasoning tasks with resource-constrained resources and for large-scale scenarios. Also, the interaction aspect is very important in smart spaces. Interaction in smart space is a very broad research topic by itself, and it is not considered in detail in this thesis. Instead, we investigate how reasoning facilitates and affects the different forms of interaction in smart spaces.

We approach the second research question mainly from the service composition point of view. First, we study how rule-based reasoning supports application composition process with a physical user interface. Second, we study how ubiquitous services can be created with rules supported by middleware.

The third research question explores the capabilities of rule-based reasoning to facilitate self-adaptation of smart spaces. This is approached from a metareasoning perspective. Metareasoning generally means reasoning about reasoning [32]. In terms of ubiquitous computing and smart spaces, metareasoning is the analysis of how well the decisions and actions of a smart space and its applications support users in their tasks. This research question seeks to explore how rule-based reasoning can be used to implement such functionality.

Research conducted in this thesis has both theoretical and practical contributions. The theoretical contribution is presented with the concepts, designed frameworks and methods. The practical contribution is presented with prototypes verifying the theoretical concepts.

1.3 Research methodology

The research presented in this thesis was conducted by following the constructive research approach [33]. Based on theoretical studies, the target problems were identified, appropriate constructs were developed, followed by their verification through prototype implementations and their evaluation.

Publication IV presents a conceptual framework. This framework was verified with the simulation of a specific smart space scenario. Verification with simulation was chosen because financial and time limitations prevented implementing a fully functioning system. Publication VI presents reference architecture which equips a ubiquitous learning system with the conceptual framework of Publication IV. Only the theoretical analysis is presented in Publication VI.

The rest of the publications contain verifications with prototype systems. They were implemented in an iterative manner. Standard quantitative and qualitative criteria were used in order to evaluate the prototypes. For instance, in Publication I, time measurements were conducted to evaluate how fast the reasoning algorithm performed its tasks. Resource consumption was measured as well. Publication II uses the amount of Source Lines of Code (SLOC), the amount of network packets, bytes load and time delay, as well as qualitative parameters to evaluate the proposed solution. Data were analysed with standard statistical measures.

1.4 Scientific contribution

This thesis consists of seven research articles as outlined in Figure 3. The thesis author is the main author in all publications, except Publication III. The work reported in the publications was done in collaboration with other researchers of the University of Oulu. The publications are presented in chronological order, Section 1.5 gives more details. The scientific contribution of articles, the role of the author and other researchers are as follows:

Publication I discusses the advantages and drawbacks of integrating a reasoning engine into a ubiquitous system, both qualitatively and quantitatively. Also, the publication reports comparison of three reasoning implementations with respect to performance and resource consumption. This work was done in collaboration with Iván Sánchez and Timo Saloranta, who were responsible for implementing the REACHeS platform, file storage system and Collect&Drop mobile client. The author was responsible for implementing the reasoning component for this system. Data collection and analysis were performed by the author of this thesis alone.

Publication II presents Perception Framework, a middleware solution for the development of ubiquitous services. This article demonstrates how rules can be used to create ubiquitous services. The work for this article was done in collaboration with a number of researchers. Xiang Su took part in the Perception Framework design, as well as in manuscript writing. Oleg Davidyuk and Jiehan Zhou contributed to manuscript writing. The author of this thesis was responsible

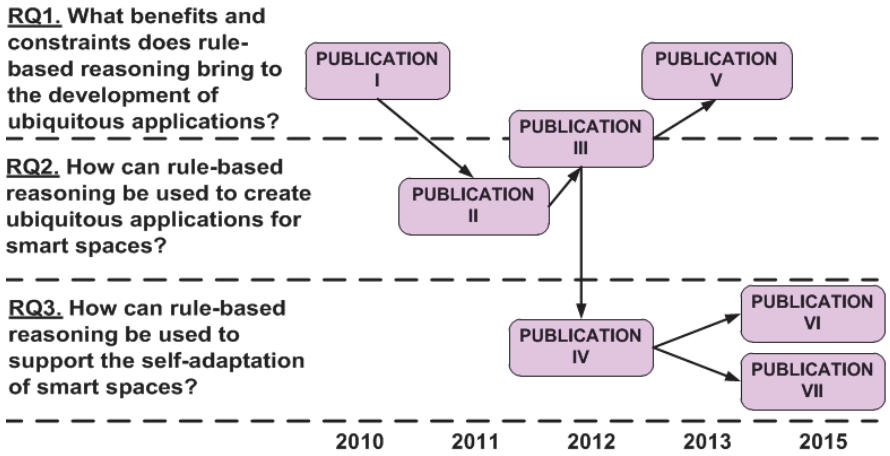

Fig. 3. Publications relations to research questions and each other.

for the Perception Framework design, implementation, testing, verification and analysis, and also had the main responsibility for writing the manuscript.

Publication III reports the design, implementation and evaluation of a prototype system for context-aware application composition. This application demonstrates the usage of reasoning to create ubiquitous applications with a physical user interface. This study was done in collaboration with several researchers. Oleg Davidyuk presented the initial idea and designed the user interface. Jussi Mäkipelto implemented the prototype. Mikko Pyykkönen made the graphical icon design for mobile clients, as well as the graphical design for the desktop application. Iván Sánchez was responsible for the REACHeS platform. Oleg Davidyuk had the main responsibility in manuscript writing, conducting the user experiment and results analysis. The author of this thesis designed and implemented the reasoning component, contributed with conducting the user experiment, reasoning-related analysis and text.

Publication IV proposes to make a distinction between context-based adaptation activities of smart spaces, and monitoring and controlling of these adaptation activities. The author of this thesis was responsible for all work reported in this article, namely for developing meta-level control concepts to smart spaces, designing the framework, creating the scenario, implementing the simulation and analysing results.

Publication V considers a smart space interaction concept. The publication categorises the types of interaction in smart spaces from the participants'

perspective, and reviews the technologies enabling and affecting the interaction in smart spaces. This article was done in collaboration with the following researchers: Oleg Davidyuk and Xiang Su took part in discussions and manuscript writing. Additionally, Oleg Davidyuk is the main contributor for the QuizBlasters case. Xiang Su is one of the main contributors for the EMA and SI cases. The author of this thesis was responsible for the theoretical background, like introducing concepts related to smart space interaction, its categorization, and identifying necessary technologies. Moreover, the author of this thesis analysed the three cases with respect to the theoretical aspects introduced and the conclusions made. The author of this thesis implemented the reasoning component for one case (QuizBlasters) and is one of main contributors for the remaining two cases (EMA and SI).

Publication VI analyses the state of the art in ubiquitous learning, and explores how the roles of learners, instructors, developers, and researchers are supported in the literature. Moreover, the article analyses how the meta-level framework presented in **Publication IV** can be implemented for ubiquitous learning applications. This article was done in collaboration with Iván Sánchez and Marta Cortés, who contributed with identification of the roles and their needs in ubiquitous learning systems and with manuscript writing. The author of this thesis was responsible for the state of the art analysis, identification of the needs, application of the meta-level framework to ubiquitous learning systems and corresponding analysis, and had the main responsibility in manuscript writing.

Publication VII partially implements the ideas of **Publication IV** in the traffic domain. The article presents reference architecture for a ubiquitous driving assisting system and exemplifies it with a concrete implementation. This article was done in collaboration with the following researchers: Anja Keskinarkaus was responsible for digital map preparation and development of scripts retrieving the route properties from the digital map; Satu Tamminen provided support for model development, as well as served as the main test user for the system; Susanna Pirttikangas took an active part in discussions and manuscript writing. The author of this thesis developed the reference architecture for the ubiquitous driving assistant system, took part in digital map preparation, developed the prototype, analysed results and was the main writer of the article.

Professor Jukka Riekki has supervised the research work presented in the articles and advised the author about the work performed and conclusions to be drawn.

1.5 Research history

The research work presented in this thesis was carried out in the Devices and Interoperability Ecosystem (DIEM)[2], Pervasive Service Computing (PSC)[3], Active Learning Spaces[4], and Data to Intelligence (D2I)[5] research projects in the University of Oulu from 2009–2015.

The research was started in the DIEM project, which the author joined in 2009. The purpose of the DIEM programme was to create concepts and implement a generic and scalable smart space interoperability solution and platform, which can be adapted to various domains and applications. In this project, the theoretical studies of reasoning mechanisms for ubiquitous computing were conducted. Particularly, research on the benefits of rule-based reasoning over a hard-coded approach led to the Collect&Drop prototype (Publication I in Figure 3).

After successful implementation of the Collect&Drop prototype, the author was interested in whether rules can be used in a larger scale scenario. For this case, a ubiquitous campus scenario[6] was constructed, as well as rules defining the application logic [34].

A considerable amount of the conducted research was performed in the PSC research project (2009–2011). The project concentrated on the core idea of ubiquitous computing, namely unobtrusive support of users in their daily activities. This project tackled this issue from a service composition perspective. At this time, the author was mostly interested in how to use rule-based reasoning as a tool to create ubiquitous services. In this project, the author designed and implemented the middleware to create the ubiquitous services with rules, namely the Perception Framework (Publication II in Figure 3). The implemented prototype demonstrated the benefits of using such middleware from the perspective of the service developer; however, the approach introduces communication delays, resulting from the distributed nature of the Perception Framework.

---

[2] Devices and Interoperability Ecosystem ICT SHOK project, funded by TIVIT, Tekes (Finnish Funding Agency for Technology and Innovation), companies and research institutions.

[3] Pervasive Service Computing: A Solution Based on Web Services, funded in the Ubiquitous Computing and Diversity of Communication (MOTIVE) program by the Academy of Finland.

[4] Active Learning Spaces, supported by Tekes under Learning solutions program.

[5] Project is supported by Tekes as part of the Data to Intelligence Program of DIGILE (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT and digital business).

[6] This article is not included as the part of the thesis, however we refer to it when discuss the scale of reasoning. This article was done in collaboration with Xiang Su and Professor Jukka Riekki.

Another prototype implemented in the PSC project is called QuizBlasters (Publication III in Figure 3). This system demonstrates context-aware application composition. Several research issues related to rule-based reasoning were explored with this system. First, reasoning tasks were performed on resource-constrained devices, such as mobile phones. Second, reasoning was distributed. Both of these aspects are considered to be very important for ubiquitous systems development.

The QuizBlasters prototype inspired further research towards the control and adaptation of reasoning facilities in smart spaces, namely, meta-control of smart space. This was inspired by the fact that smart spaces should be able to make run-time judgements about their own performance and make corrective actions if something goes wrong. First, the conceptual framework for the meta-level in smart space was designed and supported with a prolog-based simulation (Publication IV in Figure 3). The simulation revealed the possibilities and challenges for such a meta-level control for smart spaces with rules.

Analysis of the work conducted in PSC led to a better understanding of how reasoning facilitates and affects overall interaction between users and the resources of smart spaces. The author was curious about whether reasoning architecture affects the interaction type in a ubiquitous application, and how. This resulted in research regarding interaction in smart spaces, where several prototypes are compared (Publication V at Figure 3).

Further investigation of meta-level control of smart spaces is done within the Active Learning Spaces project in which the author worked for a few months in 2013. In this work, the meta-level framework is applied to ubiquitous learning systems (Publication VI in Figure 3). A more advanced study about meta-level control was conducted in the D2I project where the author worked between 2012–2015. The purpose of the project is to develop solutions, algorithms and systems for intelligent data analyses. In order to evaluate the meta-level concept, the author designed and implemented a driving coach system (Publication VII in Figure 3).

1.6 Structure of thesis

This manuscript is written as a compilation of published research articles introduced with a summary text. Hence, the summary text gives a solid ground for all the research reported in research articles and is structured as follows: Chapter 1 introduces the research topic, as well as defines the research questions of this manuscript. In addition, it describes the author's contribution. Chapter 2 gives background information, as well as surveys related work. Concrete research and

experiments are described in Chapter 3. The conducted research is finalized with Chapter 4, where the research questions are revised, open issues are revealed, as well as future work is outlined. Finally, Chapter 5 concludes the manuscript.

2 Background and related work

This chapter introduces key-concepts in the ubiquitous computing domain. Namely, we explore what is context, context-awareness, and context reasoning. Furthermore, we present rule-based reasoning in more detail. Finally, we review the related work on developing ubiquitous applications and present how it utilizes reasoning.

2.1 Context and context-awareness

Context is one of the key-terms of ubiquitous computing. However, it is difficult to define the term context generally. Early works define context with synonyms, like environment and situation, and examples, like time and location [1]. Then, researchers emphasize the aspects of context, like where you are, who you are with, and what resources are nearby [35]. The most famous definition of context in ubiquitous computing community is given by Dey [1]: "*Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*" However, even this definition has been criticised for being too general. Winograd [36] emphasizes the operational side of the context: "*Something is context because of the way it is used in interpretation, not due to its inherent properties. The voltage on the power lines is context if there is some action by the user and/or computer whose interpretation is dependent on it, but otherwise is just part of the environment.*"

Indeed, the notion of context itself is open and general, and often it is difficult to tell what is context and what is not. Moreover, what is considered as context in one setting might not be context in another setting [37]. Actually, it is not possible to enumerate a priori all important characteristics of a situation [1], hence it is important to define the scope of the context in relation to the designed system, its application domain, and usage environment. That is why recently researchers have categorized context, and defined its characteristics. For instance, Zimmermann introduces five categories for elements that describe context: individuality, activity, location, time, and relations [38]. Soylu et al. [37] give a survey about context categorization dimensions, and propose their context categories, like user context (internal, external), device context (hard, soft), application context, information context (physical, digital), environmental context, time context, historical context and relational context.

A system that "*uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task*" is context-aware [1]. That is, a context-aware application utilizes context information in order to modify and adapt its behaviour at run-time. Dey and Abowd [39] propose that a context-aware application should support the following features: 1) presentation of information and services to a user; 2) automatic execution of a service; and 3) tagging of context to information for later retrieval. Poslad [3] provides a broader view on the properties of context-aware systems. He points out that context-aware systems actively adapt to context changes in a dynamic environment, rather than just present context changes to a user. Also, in addition to user-awareness and environment-awareness, these systems must be aware of the ICT infrastructure in which they exist.

Many context-aware systems have been developed. Pioneering works include the Active Badge Location system [40], able to forward phone calls to the locations of the users; the Personal Shop Assistant [41] guiding users in a shop and providing necessary details about the items available; and the Cyberguide system [42], providing tour guide information to a user based on his location and history. More recent examples include the Event Map Application [10], giving personalised alerts based on location; and the adaptive U-learning math path system [43], a context-aware application for learning, just to mention a few. More details can be found in the surveys by Chen and Kotz [44] and Baldauf et al. [45].

Developing a context-aware application is a demanding task. Wei and Chan [7] specify four basic issues to consider when developing a context-aware system: what is the context, how to acquire the context, how to represent the context, and how to adapt to the context. Hence, generally a context-aware system consists of the following blocks: context acquisition, context modelling, and context reasoning and action, as described in the Introduction.

Context acquisition involves obtaining of the environmental (physical, social, etc.) information. This can be done with physical sensors, services, by manual input, or by learning. Often, complex pre-processing of signal data is required, like segmentation, feature extraction, and classification, in order to obtain a meaningful low-level context. Also, sometimes data from several sources have to be combined in order to get a meaningful context; this is called data fusion.

Context modelling refers to representing the part of real world that is relevant to the designed context-aware application. Naturally, this representation should be adequate, faithful and allow efficient manipulations. Wei and Chan [7] emphasize that context modelling is concerned with representing, structuring and organizing

contextual data, and the relationship between them, in order to facilitate their storage and operations. They consider three basic aspects that a well-designed context model should address: *data structure*, *integrity,* and *manipulation*. Data structure is necessary for context information exchange and facilitates information storage, validation, modification and reasoning. Integrity refers to the support of validation of both structure and actual data of contextual information. Manipulation defines the operations that can be applied to data structures for reasoning. Strang and Linnhoff-Popien [46] also introduce their requirements to context modelling approach, e.g. supporting the incompleteness and ambiguity of context, as one cannot guarantee correct and complete information from physical sensors. Many approaches exist to model the context, like Key-value, Markup Scheme models, Graphical models, Object Oriented models, Logic based models, and Ontology based models [21], [46], [47].

2.2 Context reasoning

Reasoning, also referred to as inferencing, involves logical operations on logical sentences or statements within a model (the world, the application domain) in order to draw conclusions and derive other sentences [3]. The term of reasoning as well the usage of reasoning techniques and algorithms are widely applied in the field of ubiquitous computing. Nurmi and Floréen [2] define context reasoning as deducing new and relevant information for the use of applications and users from the various sources of context data. Hence, context reasoning is reasoning where logical sentences and statements describe context. Basically, context reasoning supports decision-making by transforming the acquired contextual information about the environment and users to possible actions.

As was mentioned in the Introduction, context reasoning can serve for multiple purposes in ubiquitous computing [8]. First, reasoning deals with context data verification and validation, and hence, with system consistency support. This is required because context providers, such as sensors, are error-prone, and the contextual data they provide can be uncertain, unknown, or even wrong. The second purpose is inferring high-level context, that is, knowledge discovery from implicit context information or low-level context. For example, from the low-level context obtained from sensors “light is on”, “microphone is active” and “projector is active”, we can infer the high-level context “there is a meeting in the room”, which is more informative and can guide a context-aware application more

precisely. The third purpose refers to triggering adaptive actions from the system, based on context changes.

Table 1 lists some approaches used in the ubiquitous computing domain to implement context reasoning. The interested reader is referred to surveys [11],[12],[21],[48] for more details. Some approaches, like rule-based reasoning and ontological reasoning, use logic formalisms. To handle information that is unknown at the design-time, machine learning methods are used (neural network, decision tree, case-based reasoning). Probabilistic approaches cope with uncertainty (Hidden Markov Models, Bayesian network). There are also other aspects which can be considered, like support for time. It is clear that some approaches suit some tasks better than others. For instance, user preferences can be well represented with logic-based formalism, however if no preferences are known, they should be learnt first. In ubiquitous computing applications, often more than one approach is applied to implement reasoning, for instance ontological reasoning is quite often used together with rule-based reasoning.

As can be seen from Table 1, there are many alternatives for implementing reasoning for context-aware applications. Also, reasoning mechanisms are tightly connected with context models, as generally a formal representation model defines the mechanisms which can be applied to it [11],[12]. For instance, ontological reasoning includes inferencing for Description logics (DL)-based representations. Such representation allows describing the application domain at the conceptual level first (its terminology, i.e. TBox), and then by using this terminology, to name individuals functioning in this domain (ABox) [49]. The basic TBox operations are determining whether a description is satisfiable (i.e., non-contradictory) and whether one description is more general than another one. The main operation for ABox is finding out whether its set of assertions is consistent. This means entailment that a particular individual is an instance of a given concept [47], [49].

Case-based reasoning (CBR) is an artificial intelligence technique that allows solving new problems by analysing the solutions of previous, similar problems. Hence, this technique is the one which uses experience for decision-making. A case can be defined as a problem and its solution. All cases are stored in a case library (case base). Finding a solution for a problem includes the following steps: 1) the case base is looked up for the closest case, 2) the solution of the retrieved case is applied to the initial problem, 3) the solution is revised and if needed, 4) the solution is added as a new case for possible future problems [51].

Table 1. Examples of context reasoning in ubiquitous applications.

Technique	Benefits	Weaknesses	Usage examples
Rule-based reasoning	- Simple definition - Simple interpretation by the users - Easily extendable	- Difficult maintenance and verification - No built mechanism to deal with uncertainties, except encoding probabilities into rules	- Defining higher-level context - Defining actions based on conditions - Defining policies and preferences
Ontological reasoning	- Thorough concept definitions - Gives possibilities to reason about objects and their relations	- Not evident how to handle uncertainties, however some efforts exist [50].	- Defining relationships - Validation and verification - Often ontological reasoning alone is not enough for ubiquitous systems and rule-based reasoning is required
Case-based reasoning	- Deals with unknown problems	- Careful specification of what are cases and what is their similarity are required	- Defining actions based on certain situation - Diagnostics tasks
Neural network	- Deals with complex problems which may have difficult formalization	- Requires massive data to learn the model - Requires knowledge for parameters adjustments (number of layers, neurons)	- Defining the low-level context from signal data
Decision trees	- Uncovers the relations between inputs and outputs - Can be used for both classification and regression tasks - Easily interpreted by the users	- Complexity to create the large decision trees - May be difficult to create one if many different interrelations exist in the data	- Defining the actions based on certain situation
Bayesian network	- Deals with uncertainty of context	- Requires massive data to create the model (learn prior probabilities)	- Data fusion, - Situation recognition
Hidden Markov Models	- Deals with uncertainty of context - Considers the timeline	- Requires massive data to create the model (learn transmission probabilities)	- Situation recognition

Probabilistic context reasoning deals with imperfect context information obtained from sensors, physical devices, and users. Henricksen and Indulska [9] provide four types of context information imperfection: *unknown* (no information about the property is available), *ambiguous* (several different reports about the property are available), *imprecise* (reported state is a correct yet inexact approximation of the true state), *erroneous* (mismatch between the actual and reported states of the property). Several approaches can be used for reasoning with uncertainty, for example probabilistic logic, fuzzy logic, and Bayesian networks.

Often, hybrid systems are used where different approaches are combined, like it is a common practice nowadays to use ontological reasoning with rule-based reasoning [47]. Also, probabilistic models can be combined with other forms to support context ambiguity [52],[53]. More discussion on the reasoning mechanisms can be found in [11],[12],[21].

2.3 Rule-based reasoning

Rule-based reasoning employs the power of rules, which can be used to express policies, preferences, and constraints. Also, rules can describe high-level contexts, as well as contain adaptation information. Bikakis and Antoniou [54] point out that rules are simple, flexible, formal, expressive, modular, and facilitate high-level context abstraction and integration with ontology models.

Rule-based reasoning is built from facts and rules. Facts are unconditional statements that are assumed to be correct at the time they are used [53], so facts are descriptions of a situation. A collection of facts forms a fact base, which is used as data input for rule execution. A rule can be defined as an instruction for an action that can be applied in a certain situation. Rules are commonly represented as follows: *If* <*Condition*> *then* <*Action*>. Here, the <*Condition*> part (antecedent) represents the conditions necessary to be true for rule execution. Several conditions can be combined into a composite one by means of logical operators. The <*Action*> part (consequent) represents the derived facts following from the rule, or actions which should be performed if the rule is executed. The rule is executed only when the antecedent part is true. Facts and rules can be written in different forms of logic, like First Order Logic (FOL). This allows the application of the foundations of logical formalisms to actual inference.

A general rule-based reasoning system consists of the following core components: Rule base, Working memory, and Inference engine, see Figure 4. The *rule base* contains all the rules defined for a specific system in the form of if-then statements. These rules are provided either by a system developer, domain expert, or system user, or could be learnt by the system automatically. The *working memory* contains all the relevant facts known by the system during an operation. The *inference engine* is responsible for executing the rules whose conditions are supported by the facts of the working memory. Generally, inference engines use the following strategies: forward chaining and backward

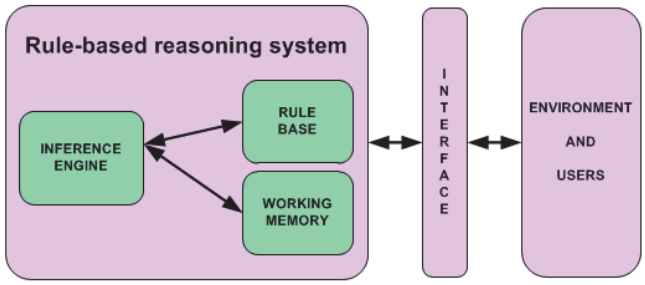

Fig. 4. Components of a general rule-based reasoning system.

chaining. Forward chaining means that the inference engine executes rules whose conditions are satisfied with facts. Different conflict resolutions strategies have been developed to cope with cases when conditions are matched for conflicting rules [3],[55]. Once executed, a rule can modify the working memory and satisfy other rules of the rule base. Hence, the procedure of rule selection for execution needs to be performed again. This process continues until no rules can be executed or reasoning is manually stopped. Backward chaining means that the inference engine operates backwards from consequents of the rules matching the goals to antecedents of these rules, checking if they are supported by facts. Also, a hybrid approach has been suggested, when a rule dependence network is built prior to running the system [53]. Such a network tells the dependencies for each rule, i.e. which other rules may enable it, and which rules it may enable.

The *interface* provides communication facilities between the rule-based reasoning system and the outside world. The facts and rules used by a rule-based reasoning system are supplied by the users and environment. For instance, a temperature sensor could send a measurement to the system as the fact of the current temperature in a room. On the other hand, a rule-based reasoning system delivers the results of the inference process to the application software components, to users via GUI, or to objects of the environment. The environment and the users are the main knowledge suppliers and knowledge consumers of rule-based reasoning system.

In order to utilize rule-based reasoning, one has to present the knowledge in terms of facts and rules, and implement the interfaces. The inference engine will take care of all the rule execution steps. This makes rule-based reasoning systems different from conventional programs entirely written in procedural language. Rule-based reasoning systems follow the declarative programming approach, as domain knowledge is presented with facts and rules without any explicit

specification of the control flow. A separate entity, inference engine, interprets the rules and controls their execution [13], [55]. Conventional programs do not demonstrate such clean separation. This means that declarative knowledge specified with facts and rules can be easily modified and replaced without influencing the inference engine. At the same time, developers should pay attention to rule base design, in order to achieve consistency, to avoid redundancy, and to reach completeness [13]. This is required, as facts and rules constitute the main component defining the system behaviour. To conclude the introduction about the rule-based reasoning, we outline the advantages and drawbacks of such systems with Table 2 (reworked from [55]).

Table 2. Advantages and drawbacks of rule-based reasoning systems.

Advantages of rule-based reasoning	Disadvantages of rule-based reasoning
Simplicity and explanation capabilities– rules can be easily interpreted by developers and users. Additionally, it is possible to trace the rules to figure out the cause-effect relations.	Requires thorough domain formalization in design time – this means that all the domain information should be encoded in order to retrieve adequate decisions. However, integration with machine learning algorithms overcomes this issue.
Modularity – each rule encodes the unit of knowledge and can be easily inserted and removed from the knowledge base. This also implies some form of independence from other rules; however, in practice relations between rules should be considered.	Difficult maintenance – it is difficult to analyse and maintain large rule bases, as there is limited tool support.
Rule-based reasoning systems allow the separation of the knowledge from other functional components of the system. Such isolation leads to easier application support.	Additional external components may provide delay issues for the overall system, especially if it is a highly distributed system.
Easy integration – rule-based representation can be easily integrated with other context models, e.g. it is common practice nowadays to integrate ontological models with rule-based reasoning [54].	
Inference engine solutions are available for both resource-rich platforms, as PCs, as well as for resource-constrained devices, like mobile phones.	Inefficiency – comes from the fact that many steps should be performed by the inference engine for each executed rule; however, new matching algorithms may improve this situation.
Interoperability – research initiatives exist which propose the rule-transfer mechanisms from one reasoning system to another one [56].	Lack of reusability- means the lack of open and standardized rule sets available for application domains, hence every time the developers start from the scratch.

2.4 Reasoning to implement application functionality

Reasoning has become a natural and necessary part of ubiquitous applications. Every ubiquitous application needs to process the incoming context information to provide its adaptive functionality. Three main responsibilities for reasoning, such

as dealing with imperfection and uncertainty of context, defining the higher-level context, and triggering adaptation actions for certain situation were identified in the Introduction of this thesis. This section reviews related work which utilizes rule-based reasoning mechanisms for these functions, and hence uses rules as the system component (Figure 2, bottom).

Even though rule-based systems have some drawbacks they are still among the most popular reasoning components in ubiquitous computing domain. Lim and Dey [57] reviewed 114 context-aware applications from the major ubiquitous computing conferences (CHI 2003–2009, Ubicomp 2004–2009 and Pervasive 2004–2009) and showed that more than 50% of analysed works utilize rules for reasoning. A more recent study analysing 50 context-aware projects performed in 2001–2011 also demonstrates the significance of rules [21].

Rules are mostly used to define the ***higher-level*** context and to describe the ***context-aware adaptations***. Higher-level context specification is required to create a hierarchical structure between elementary contexts, moreover, higher-level context may provide more meaningful information or specifications to the applications. For instance, from elementary facts that a person is lying in the living room, where the lights are off Cao et al.[58] infer that the person's activity is sleeping, hence services tailored to this activity could be activated. One may find similar rules for other domains as well, e.g. in ubiquitous healthcare [59]. Another proposal for situation detection based on rules is presented by Costa et al [60]. Also, additional rules can be formed to specify the situation. For instance, Toutain et al. [61] exemplify the transitivity rule with respect to a person's location. Namely, if the person is inside a place which is inside a larger place, then the person is inside the larger place as well. In turn, Bonino and Corno [62] utilize rules to define structural and state properties of intelligent environments, which contain a variable set of home appliances.

Describing situations leads to their use for context-aware adaptation. For instance, defining that a person is sleeping may lead to certain actions being performed, like switching the mobile phone to silent mode, activating a morning alarm, etc. Rules suit very well for formulating the actions to perform in a situation. Su et al. [10] define rules to remind users about tasks to be performed based on location and time context. For instance, a system may remind the parent closest to school to pick up their child. Another recommendation system is suggested by Armenatzoglou et al. [63] where alerts and route recommendations are presented to a user based on his profile information and location.

One of the main use of rules is to describe ***policies and preferences*** to personalize the services and devices [64]. For instance, with Microsoft on{X}[7] one can create rules to personalize an Android phone. To personalize applications, Fong et al. [65] propose using defeasible-logic rules to describe system behaviour. This actually raises other issues, such as providing users with the tools to encode their preferences. For instance, Garcia-Herranz et al. [66] propose a rule-based language to describe preferences for controlling smart environments. Another approach is suggested by Boyaci et al. [67]: a language and infrastructure are designed for controlling a smart environment. This SECE system supports different kinds of rules, like location-based rules, time-based rules, calendar-based, communication and context-based rules. A distinguishing feature of this proposal is its English-like syntax for users. Another approach is to learn user preferences from the actual use of the system, as suggested by Papadopoulou et al. [68] who used the C4.5 algorithm to figure out user preferences in the form of rules from collected usage traces.

For a user, it is very important to understand what is happening in the environment, why it is happening, and how to control and modify system behaviour, especially if user preferences were not inputted by the user, but learned by a learning algorithm. Due to their expressivity, rules are useful ***to explain users*** the system behaviour, as suggested e.g. in PervasiveCrystal [24], Lim and Dey [57], and Fong et al. [65].

A rule-based reasoning approach can be used to deal with ***imperfection and uncertainty*** of context. The simplest approach is to create for the raw signal preprocessing rules that define operational limits for an entity. This simplest approach might be useful as a preliminary filtering step [69], but different types of sensors and entities require tailored strategies [70]. Another approach is suggested by Degeler and Lazovik [71]. The authors propose a data structure, called a context consistency diagram (CCD), for efficient tracking of sensed contexts. The idea is to use the rules of sensor dependencies to figure out faulty sensor measurements. For instance, a TV can be on only if electricity exists. If other appliances are off and there is no electricity, a sensor reporting that the TV is on is probably faulty. Similarly, an approach based on context-dependencies is used by Filho and Agoulmine [72]. Their approach is designed to overcome context conflicts resulting from faulty measurements. For instance, if two different kinds of sensors report a different location of a person (e.g. GPS and calendar record), which one to be

[7] URI: https://www.onx.ms Cited 2015/05/11

believed? The different forms of context inconsistency are addressed by Lee et al. [29]. The authors support context consistency within dynamic context changes, e.g. untimely (early or late) elimination of invalid contexts, and ignoring the co-existence relationships among deduced contexts. Their context elimination rules describe the semantics of context invalidity to solve context inconsistency issues.

From the architecture point of view, rule-based reasoning components of ubiquitous applications are mostly centralized, following the client-server application model [10],[63],[58],[59]. This means that a specific reasoning component in the system queries the data required for decision-making. The reasoning result is then sent to the interested components. For many types of ubiquitous applications, the centralized approach works well, e.g. for reminder systems. However, a distributed approach may provide system modularity, efficiency, and privacy support to ubiquitous applications [8], [48]. Researchers in the ubiquitous computing domain have recently started to consider distributed reasoning. With distribution of a reasoning component, we refer in this thesis to distributing the knowledge base and inference rules between system components based on their resources, location and policies. There are certain challenges when designing such a system, like knowledge representation and reasoning coordination [73].

Rule-based reasoning provides facilities for distributing reasoning. One approach is to determine hierarchy for the rules and assign rules to components based on this hierarchy. For instance, Gu et al. [74] suggest grouping peers into semantic clusters, hence queries are sent only to the peers of a dedicated cluster. Another approach is to distribute independent rule sets to the different nodes of the system [8],[75],[76]. This approach allows application components to use their local knowledge bases and operate with the information they are interested in. Such approach can be met in agent-based and P2P systems. For instance, Gross et al. [77] utilize a P2P network and abstract the access to sensor data using rule-based inference. Each peer in their system has a local rule-base and communicates with other peers with rule-based queries. An interesting approach is suggested by Bikakis and Antoniou [78], they suggest using Multi-Context Systems together with defeasible logic for their autonomous agents. Basically, their agents possess the local knowledge bases as well as mappings to knowledge bases of other agents in the form of bridge rules. Together with implementation of algorithms for distributing reasoning, the format to exchange the rules between different systems is under development [56].

2.5 Reasoning to create applications

The creation of ubiquitous applications is a demanding task involving many technical and design challenges. Technical challenges include high diversity of communication and computational resources, as well as high mobility of users. Design challenges include defining the context, and the means to acquire and formalize it, as well as solutions for context usage and adaptation to it [7]. We can identify three ways of creating context-aware applications: application creation from scratch, application creation by composition, and application creation with middleware support.

Application creation from scratch. Creating ubiquitous application from scratch still seems to be the case for many context-aware applications. This approach involves a large amount of work, and it is both time and resource consuming. However, this approach gives the freedom to select the technologies and methods for both the design and implementation. This thesis does not provide further details about this approach.

Application creation by composition. The second approach to creating a ubiquitous application is by means of composition. In fact, many ubiquitous applications can be considered as compositions where different software components, Web services, and computational devices are connected together to provide a level of intelligence and functionality which cannot be achieved by one isolated component. Application composition is a broad research topic itself [79], [80] and in this thesis it is described very briefly. In this thesis, application composition means building applications by assembling them from other services (third party or offered by computing devices) offered by the smart space [81]. A simplified and general framework for application composition is presented with the Figure 5.

All available services should upload their descriptions to the uniform repository, so that the discovery mechanism would be able to locate them. This description specifies the functionality of the service, interfaces to execute this functionality, as well as requirements and data flow. WSDL [82] and OWL-S [83] serve as examples of service description formats. The composition engine is responsible for the creation of the composite application, based on the given criteria or specifications which come from the composite service developer or user. With the help of service discovery, the composition engine is able to find the services satisfying the given criteria and, by means of a composition algorithm or guided by a user, it creates the composite application specification. Actually,

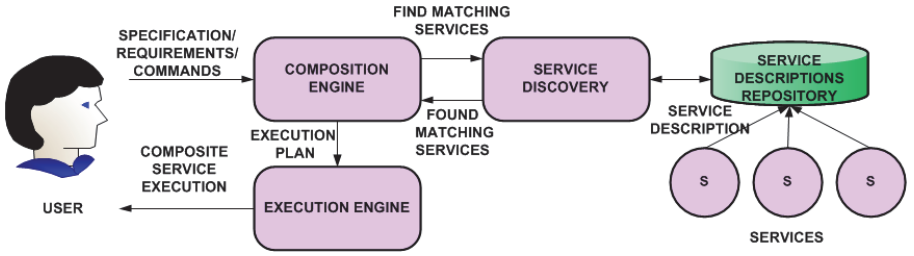

Fig. 5. General application composition framework.

several composition applications can be constructed, but only the best one (according to the criteria) is selected for execution. The execution engine gets the composite application specification and controls the execution of the composite service that can also be modified at run-time (i.e. adaptive service composition [84]).

A number of composition approaches exist: static and dynamic, model-driven, business rules driven and declarative, automated and manual, semantic and syntactic composition [85]. Also, different mechanisms exist for the composition engine to select and map the existing services to fulfil the required criteria or specification, for example AI techniques can be used [86].

By its nature, a smart space should provide opportunities to its inhabitants to create their own composite applications, and customize and personalize existing ones. This requires end-user involvement in the application composition process. For instance, Zhou et al. [87] use a rule-based approach to select appropriate services to be included into a composite application execution plan. Services are selected based on user preferences and current calendar information which constitute the context in their application. In fact, in this proposal, application composition is invisible for the user, who just specifies the criteria. Also, Semantic Web technologies are proposed for automatic composition of the ubiquitous applications [88], [89]. This type of composition where the user is involved at a minimum in the composition process is called automated composition [79].

In the proposal by Xiao et al. [19], users are more involved in the composition process. Services are abstracted as tasks with descriptive key-words, so end-users can use these key words to locate the desired services. Basically, a composition is presented as a sequence of tasks, which users are able to edit. Such composition involving end-users is called semi-automatic or interactive composition [79]. Zahoor et al. [90] demonstrate rule-based Web service composition guided by users who manually specify the composition flow using a user interface. Similarly,

Beattie et al. [91] suggest a user interface to create rules to support self-management in dementia through remote monitoring of activities within a smart home environment. Mavrommati et al. [92] and Wisner and Kalofonos [93] suggest a holistic infrastructure together with a graphical editor to compose ubiquitous applications utilizing different proprietary algorithms. This type of application composition, where users are supplied with graphical tools to compose an application, is called visual-based application composition [79].

Another interesting approach to composing ubiquitous applications by end-users is by means of physical interactions with the resources of a smart space. For instance, Davidyuk et al. [81] suggest a touch-based user interface to compose applications. Users can build a composite application by physically selecting required resources or service abstractions by utilizing NFC technology. Proprietary evolutionary algorithms are used to implement application composition. Chin et al. [94] suggest an approach where end-users show to the system the required behaviour by natural physical interaction with resources of the environment. In this proposal, the actions of the user in the environment are encoded with rules. A related approach is suggested by Gajos et al. [20] where end-users compose applications by instructing a smart space with a sequence of actions; also by interacting with the resources of the smart space. These approaches demonstrate application composition fully performed by end-users. Davidyuk et al. [81] refer to this type of composition as manual, where users are fully responsible for the selection and the workflow of all services. More advanced approaches involving learning mechanisms would allow discovering of the behavioural patterns from interactions between a user and the resources of the smart space [95].

As a summary, ubiquitous applications created by a composition mechanism require intelligent solutions for both performing the actual composition and implementing the adaptive behaviour. Different composition mechanisms are presented in related work. Generally, a semantic description of services (like OWL-S) offers better expressivity, as well as easier integration and reasoning capabilities [88], [19]. The most common abstractions to represent a composite application are tasks with goals [88], templates, and general workflows [90]. This is natural, as it is convenient for users to decompose large tasks into smaller parts and define how these parts are related to each other. Also, goal-based thinking abstracts such difficult concepts as “service input” and “service output” which link services. However, defining goals and data flow can be challenging for end-users. For this reason, recording approaches have been suggested, where the system records interaction with a user and the resources of a smart space and executes the recorded

sequence when required [94], [20]. This approach is based on the assumption that users know by themselves their goals, desires, as well as the basic functionality of the resources of a smart space. However, this approach makes it challenging for a user to define context-based behaviours, as well as define less common data flows. Different approaches are suggested for performing the actual composition like rule-based [90], evolutionary algorithms [81], other proprietary solutions [92], [93]. Rule-based composition is used to represent both service constraints [90] and adaptive control flow [87]. Also, rules are considered convenient for encoding the actions performed by users [94], [95].

Application creation with middleware support. As almost every ubiquitous application requires context acquisition, context modelling, and context reasoning components, the logical step to reduce application development efforts is to build a middleware offering these components for several applications. The middleware deals with the underlying infrastructure and provides reusable software components and application interfaces for end-applications [96].

Different approaches have been suggested to classify the middleware for ubiquitous systems. For instance, Schiele et al. [97] suggest the following three categories: the organization model, the level of abstractions, and the supported tasks. Raychoudhury et al. [98] propose analysing the middleware from the programming abstraction, system architecture, and system services and runtime support perspectives. Some works suggest more detailed and less abstract analysis of available middleware solutions, like inspecting design approaches, system architecture, communication technology, and context models [45],[99],[100]. This thesis briefly reviews the related work with focus on context modelling and reasoning technologies and mechanisms available to create ubiquitous applications. For more thorough surveys about the middleware for ubiquitous computing, the reader is referred to [21],[98],[99],[101],[22].

The first middleware proposals include the Context Toolkit by Dey [102], Gaia by Román et al. [103], and frameworks by Schmidt [104] and Mitchell [105]. These distributed architectures support the development of context-aware applications with a layered approach. In this approach, components belong to different layers, each responsible for a common task. For instance, lower-layer components retrieve raw data from sensors and deliver it to higher-layer components. Higher layer components transform the retrieved information to more abstract entities and forward it further. The highest layer is presented with actual applications. The middleware components can be shared between different applications [102],[104],[105]. Gaia [103] middleware supports both development and

execution of applications for smart spaces. The middleware manages the resources and services of a smart space, provides location, context and event services, as well as stores the information about the space; hence it provides greater functionality in comparison to other proposed solutions. The Context Toolkit encodes the context with a key-value schema [102]. Schmidt [104] tested different AI techniques to describe context, e.g. linking clusters obtained with the Kohonen Self-Organizing Map (SOM) to context labels. Mitchell [105] used the Object oriented model to present context which is then accessed via an API. Zimmerman [38] emphasizes end-user involvement in maintaining ubiquitous systems, for example, in correcting system behaviour when something goes wrong. Attribute-value pairs are used as context representation. A rule-based system determines the behaviour of the entire context-aware application.

Slightly different approaches are suggested by Chen [106] and Korpipää [107]. The core component of CoBrA architecture by Chen is Context Broker, which is responsible for all context management tasks; hence, each smart space is assumed to have such a component. Context Broker is also responsible for maintaining the consistency of the centralized context model, and for user privacy support. CoBrA uses an ontology-based context model and supports two kinds of context reasoning. The first kind of reasoning interprets sensing information, and the second detects inconsistencies. Both rule-based and ontology-based reasoning is implemented. Korpipää [107] proposes a blackboard-based architecture to create context-aware applications for mobile phones. Context Manager, the central node of this framework, is conceptually close to Context Broker in CoBrA. Context Manager stores context received from any source, and serves it to clients. A distinguished feature of this proposal is the customizer component, which allows users to define their own context-based actions for mobile phones. A specific ontology structure is used to represent the context. Context-based actions of an application can be written as scripts which are evaluated by a specific Script Engine.

A service-oriented paradigm provides convenient instruments for developing context-awareness middleware [108],[109],[110]. Such middleware is presented as a network of interconnected services. Bardram [108] suggests JCAF middleware which is based on the idea of dividing context acquisition, management and distribution in a network of cooperating services. Hence, cooperating services can query each other for context information. JCAF employs an object-oriented approach to model context. Gu et al. [109] present another service-oriented middleware, SOCAM. In SOCAM, a context model is presented with ontology, and both ontological and rule-based reasoning are possible. CoWSAMI [110] is a

middleware supporting context-awareness in ambient environments. These environments consist of mobile users and context sources that become dynamically available as users move from one location to another. CoWSAMI employs a relational-based approach to model context. Context rules are employed to provide mappings that specify how to populate context relations, with respect to the different context sources that become dynamically available.

Context-awareness middleware for mobile systems has recently attracted considerable attention [111],[112],[113],[114]. Similarly to Schmidt [104], context acquisition, processing and reasoning are located on mobile phones. Different approaches have been suggested, for instance LoCCAM middleware [113] uses an approach close to the blackboard approach by Korpipää [107]. In LoCCAM, applications are notified when the context is available in tuple space. Here, context representation is based on the attribute-value approach. Paspallis and Papadopoulos suggest a component based mobile middleware [112]. The authors separate context providers from context consumers, and these could be developed independently, and dynamically composed with the suggested middleware. A plug-in solution is used, where context consumers register their interests in certain context types, and the middleware chooses which context provider plugins should be activated. Such match finding is facilitated with the context model, either ontology or object-based. A similar plugin solution is proposed by the AWARE framework [114], which supports acquiring, sharing, and reusing a mobile context. Dogdu and Soyer [111] employ the RESTful paradigm in their layered context-aware middleware for mobile phones.

The Internet of Things (IoT) paradigm has started to demand scalable solutions due to the high amount of sensors [21]. Forsström and Kanter [115] propose peer-to-peer application architecture for pervasive IoT applications based on the open source P2P IoT platform called MediaSense. Each entity (like person, thing, location) has a digital representation of its current situation. Internal algorithms are used to locate and disseminate context. Developers are able to extend the platform with application-specific algorithms and reasoning facilities. Teixeira et al. [116] follow a service-oriented approach which heavily relies on semantics to describe devices, their data and physical attributes. The authors suggest a composition process that builds a service graph producing the desired outcome. The authors also stress use of mathematical models for service discovery and composition.

As a summary, pioneering works [102], [104], [103], [105] established the main principles and requirements for middleware for ubiquitous applications. Context specification, separation of concerns and distributed communication are

examples of requirements for such middleware [102], [105]. All of these proposals clearly separate components by their functionality, interconnect them in a hierarchical way and provide means to create ubiquitous applications which use these components. One characteristic feature of the first middleware proposals is a clear separation of concerns. Also, these architectures do not provide reasoning components, although it is possible to implement reasoning functionality to some components, e.g. to Interpreters [102]. Later proposals follow the early principles and consider higher-level functionality, such as control and manipulation tools for users and consistency support [38], [107]. Also, use of separate reasoning components becomes more evident [106], [38].

Mobility support is emphasized in ubiquitous domain research. Recent proposals develop further the concepts established by the early works [105],[104]. A distinguished feature of context-aware middleware for mobile phones is computation reflection capabilities [112],[113],[114]. Also, architectures do not provide reasoning components, although application developers or developers of context providers should define contextual relations and adaptations. Also, highly scalable solutions have been recently proposed to meet the requirements of the IoT paradigm. Finally, all the proposed middleware solutions differ in details, but their view on context management and use are similar conceptually.

2.6 Reasoning and self-adaptation in smart spaces

Smart spaces are comprised of ubiquitous applications, heterogeneous devices, technologies, and users with different needs, skills, and interests. Hence, in order to provide adequate long-time user support, such systems should be able to monitor their own state and context, detect changes and react to them (like device failure). Such capabilities are called self-adaptive, autonomic or self-management properties [27],[28],[117]. A system is self-adaptive if it is able to modify its own behaviour or structure during run-time. In this thesis, we use the self-adaptive, autonomic, and self-management terms interchangeably.

Self-management functionality is achieved with so called self-* properties: self-configuration, self-optimization, self-healing, and self-protection [118]. Self-configuration property refers to the capabilities of the system to configure itself to the needs of the user and available technological appliances. Self-optimization means that the system can tune itself to optimize performance, e.g. resource use. Self-healing refers to the capabilities of the system to identify and solve the problems raised, for instance to handle device failure. Finally, self-protection

means that the system can protect itself from different attacks and harmful use by users.

In order to achieve such properties, IBM proposed the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop reference model (Figure 6) [118]. In this model, autonomic systems are presented as interactive collections of autonomic elements. These elements are individual system constituents that contain resources and deliver services to humans and other autonomic elements [118]. In this reference model, a managed element represents any resource that is given autonomic behaviour by the autonomic manager. The autonomic manager monitors the managed element and its external environment, analyses this information based on available knowledge and plans and executes adaptations.

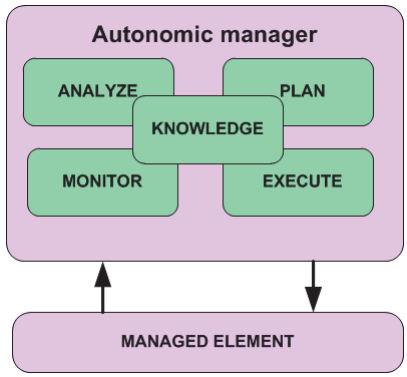

Fig. 6. IBM's MAPE-K reference model, modified from [118].

Different autonomic computing properties have been addressed by the ubiquitous computing community. For instance, Ahmed et al. [119] propose a solution to achieve self-healing services with their ubiquitous computing middleware. Trumler et al. [120] propose a P2P based middleware solution which facilitates the assembly and operation of ubiquitous systems and considers self-configuration, self-healing, and self-optimization. This is achieved by extensive monitoring of the services, resources, and the complete system at each node. Lupu et al. [121] propose a Self-Managed Cells (SMC) architectural pattern which facilitates easy addition and removal of components, handles their failures, and automatically adapts to the users' activities, environment, communication capability and interactions with other SMCs. This functionality is achieved by the use of Event Bus, Service Discovery, and Policy Management components. This solution supports self-configuration

capabilities via policy management. The self-configuration property is also addressed by Gouin-Vallerand et al. [122], whose middleware handles the tasks of application deployment and management, application configuration and application integration to the smart space. Zhang et al. [123] propose a 'LinkSmart Three Layered architectural style' to enable self-management at an architectural level. The authors use Semantic Web technologies to achieve context-awareness, genetic algorithms for configuration optimizations, and a planner for planning procedures to achieve the optimum configuration.

Some self-adapting approaches substitute the functionality suggested by ubiquitous applications when the context does not meet the requirements to provide this functionality with a sufficient QoS [124]. This fits to artefact and granularity aspects, as well as weak and strong adaptation of the adapting object in the taxonomy of self-adaptation proposed by Salehie and Tahvildari [27]. Such an approach requires mechanisms to specify the functionality, its importance and dependencies, as well as context requirements for the desired QoS. A conceptually close approach is suggested by MUSIC middleware [125]. MUSIC middleware facilitates self-adaptation of mobile and ubiquitous applications. The MUSIC adaptation process is based on an application architecture model annotated with information about application capabilities and context dependencies. MUSIC applications are built as component frameworks setting constraints on the components and their connections in a composition as well as on used external services. Hence, the middleware is able to construct multiple variants with different properties from the same application model to replace the component with better QoS characteristics on the fly, or even to construct the best application variant for the given context. A utility function is considered as criteria for adaptation. Such a utility function is provided by an application developer, and it describes how well a given application configuration fits the given context. Gunasekera et al. [126] suggest an agent-based framework for ubiquitous computing applications. This framework is centred on a component-based agent architecture that allows agents to dynamically share components with peer agents, and to adapt, based on contextual needs. Application-specific functionality of such agent is provided in form of reusable software components. Hence, agents are able to attach/detach capabilities to form composite functionality.

A different approach is suggested by Lasierra et al. [127]. The authors suggest an ontology-based approach to information management in home-based scenarios. They implement the MAPE cycle in their HOTMES ontology by using OWL and SPARQL. Management profile is one of the key concepts in this work.

Management profile is a collection of tasks that should be performed in the home gateway to provide a management service. According to the MAPE cycle, a management profile is composed of four sections: monitoring task, analysing task, planning task, and execution task. Management procedure is undertaken by handling data according to the tasks described in the management profile.

Ubiquitous computing is focused on users; hence dynamic personalization is one of the desired properties for self-managed systems that are deployed for a long time. Such property can be implemented with dynamic updates of user tasks, preferences, and profiles. There are two solutions available: explicit and implicit personalization. Explicit personalization requires active user participation in the form of specifying and updating the user tasks, profiles and preferences [128]. This solution requires carefully defined formal models, as well as a planning mechanism to match the required criteria with available system functionality. Implicit personalization does not involve user attention, but senses and analyses the actions and feedback of a user in order to build a user profile, and update it if required [129]. Such an approach usually requires machine learning mechanisms to discover dependencies between user actions, the context of these actions, and the system itself. However, such self-adaptation should be done with care, as this may lead to users feeling a loss of control [57]. Evers et al. [130] propose including the user in the adaptation. The authors argue that for the adaptation middleware it is hard to detect when it is safe to adapt, e.g. to avoid user distraction and loss of control. They introduce the user focus concept, which represents the current intention of use and comprises a collection of associated functions and software components that are in the user's perception focus, while following his actual intention. So, the middleware includes information about user focus during decision-making for adaptation. Hence, the approach suggests restricting the autonomous adaptation of the components belonging to the user focus. To achieve this, the developer has to describe all the focus elements for the application.

All the approaches considered so far are aimed at software level adaptation, while Gamrat et al. [131] consider self-adaptation with reconfigurable architectures, like FPGA. Their architecture is based on a set of elementary computing entities named Self-Adaptive Networked Entities (SANE) implementing local and autonomous decision processes that affect their own operations. Application self-adaptation is based on its creation as a SANE network, where each node can be reconfigured based on monitored data. Rules can be used to define which adaptations are to be preferred in certain situations. One more approach is suggested by Huang et al. [132]. In their SAHA system, selected functionality is

encoded directly in the hardware. To make hardware functionalities adaptable, SAHA uses FPGA technology. This allows authors to dynamically configure hardware functions on demand into FPGA devices. SAHA includes service suppliers to interact with users, a hardware adapter to manage hardware functionalities on the FPGA device, an observer to be aware of the characteristics of user applications, and a system manager to command all the adaptations. SAHA adapts its software and hardware functions based on context and requirements for better performance.

As a summary, the proposed approaches focus on diverse aspects of self-adaptation of ubiquitous systems like self-healing [119], [120], self-configuration [120], [122], [123], and self-optimization [120], [123]. Different instruments have been suggested to achieve such properties. A ubiquitous application in the reviewed proposals is mostly considered as a composition of modules or functional units, which can be substituted on the fly [124], [125], [126]. Hence, this approach is not different from the application composition concept, and requires the same intelligent functionality as described in Section 2.6. For instance, rules or planning instruments can be used to define how to assemble an application in response to changes. Generally, such an approach may possess some mixture of application logic and self-adaptive functionality. Some approaches embed autonomous functionality in such units, like [126], [131]. This provides flexibility in implementing self-adaptation, as each unit contributes to the whole application ensemble with introspective capabilities. Semantic Web technologies and rules are useful for describing self-adaptation formally [127]. They provide interoperability properties and allow the decoupling of application logic from self-adaptation logic. Some systems address self-adaptation to better support users with their changing context. Such functionality requires intelligent mechanisms to capture user feedback about system actions. Here, reasoning can be used for different purposes, e.g. rules can be used to describe user preferences or connect user preferences and context to system functionality [133]. Such information can be provided explicitly by a user, or implicitly by a learning mechanism [68].

3 Research

This thesis focuses on how rule-based reasoning can be used in ubiquitous computing. The research work conducted is grouped into three topics: study on benefits and shortcomings on implementing application functionality with rule-based reasoning, reasoning as the tool to create ubiquitous applications, and self-adaptation in smart spaces. Following sections tell in more details about conducted research under these three topics.

3.1 Research on reasoning to implement application functionality

The first research question aims to understand what rule-based reasoning brings to smart space applications in general. In order to explore this, a set of prototypes was implemented.

To explore the feasibility of implementing rule-based reasoning as a separate application module, we developed the Collect&Drop prototype described in Publication I. With this implementation we observed the quantitative and qualitative properties of such a design decision.

In order to understand the scope of using rule-based reasoning, we created a ubiquitous campus scenario [34]. This emphasizes the importance of a context model for large-scale scenarios. Rule-based reasoning with resource-constrained devices is explored in Publication III. In this publication, we present a prototype for a ubiquitous learning system, where learners use their mobile phones to interact with the system and the physical objects of a smart space.

Finally, the interaction aspect is found to be important in ubiquitous computing, and Publication V contributes to this topic.

3.1.1 Benefits and shortcomings

Publication I describes the Collect&Drop prototype, a system which uses RFID technology to store and pick digital content with mobile phones, and play it with displays and speakers of a smart space. This application implements rule-based reasoning for decision-making. The Collect&Drop system evolved from the TiPo application [134], a system which uses RFID technology to store and share multimedia content. The TiPo system implements decision-making with a hard-coded approach, whereas the Collect&Drop system dedicates decision-making to rule-based reasoning, supported by three reasoning engines: Jess, CLIPS, and Win-

Prolog. The Collect&Drop prototype gives insights into the value of a reasoning engine, both quantitatively and qualitatively, by comparing it with TiPo.

Figure 7 demonstrates the architecture of the Collect&Drop application, which is composed of four parts: REACHeS [135], a database, a reasoning engine, and a mobile phone client. The REACHeS platform is a gateway facilitating communication between Internet services, environment resources, and mobile clients. The database stores resources and content. The reasoning engine implements application functionality. The mobile phone client controls the mobile GUI and sends the user's commands to the reasoning engine. The components inside the box on the left were added to the TiPo application to implement Collect&Drop. Collect&Drop can use one of the three implemented reasoning engines (Jess, CLIPS, and Win-Prolog). This allowed us to compare the performance of these engines for this particular application.

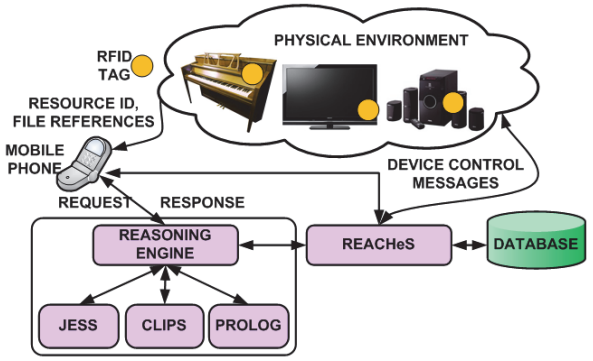

Fig. 7. Architecture of the Collect&Drop system (Redrawn from Publication I).

The Collect&Drop system defines concepts as tuples. Among others, five main domain concepts are used: User, Group, Resource, File, and Category. For instance, the GROUP(name, rights, resources) tuple identifies the group rights for resource usage. All reasoning engines use the same concepts, but for each engine they are implemented in the engine's internal language. Rules use these concepts to encode the application logic. In Collect&Drop, rules encode access policies (check if the user is allowed to use a certain resource), resource profiles (check if a selected multimedia can be executed on the resource), and resource provision (finding an alternative resource if the desired one is not available). Table 3 shows an example of the same rule in both Jess and Win-Prolog.

Table 3. Rule examples in the Collect&Drop application.

Rule description and example

Rule description

Check if the user is allowed to use the desired resource, based on the group the user belongs to, as well as the actions s/he wants to perform with the resource.

Jess rule

```
    (defrule check_user_group_rights
     "Checking if user is allowed to perform the specified action"
      ?ph<-(Phone_input (files$?file)(user_id ?id)(resource_id ?device)
                        (action ?act)(category "not defined"))
      ?us <- (User  (id ?id)(group ?group) (location ?locat))
      ?gr <- (Group (name ?group)(rights $?actions)
                    (resources $?devices))
    =>
       ( if (member$ ?act ?actions)
            then(if (member$ ?device ?devices)
                 then
                    (assert (User_message (user_id ?id)(status "OK")
                                          (message "OK")(files ?file)
                                          (resource_id ?device)))
                 else
                     (assert (User_message (user_id ?id)
                                           (status "FAILED")
                                           (message  (str-cat  "User  does  not
have rights to use the device " ?device)))))
            else
               (assert (User_message (user_id ?id) (status "FAILED")
                                     (message  (str-cat  "User  does  not  have
rights to perform an action " ?act " on the device " ?device)))))
       (retract ?ph))
```

Win-Prolog rule

```
check_user_group_rights(User_id,Resource_id,Action,Result,Message):-
  phone_input(Files,User_id,Resource_id,Action),
  user(User_id,Group,_),
  group(Group,Rights,Resources),
  (member(Action,Rights)->(
     member(Resource_id,Resources)->(Result='OK', Message='OK');
                              (Result='FAILED',
                               Message='User does not have rights to perform this
action on the device'));
                               (Result='FAILED',
                               Message='User  does  not  have  rights  to  use  the
device')).
```

The Collect&Drop system was evaluated both quantitatively and qualitatively. The quantitative evaluation was based on the performance of the Collect&Drop system compared to TiPo. Performance was analysed by measuring the response time for a varying number of smart space resources for two cases: 1) a resource is available for a user, and 2) the resource is not available, hence an alternative resource should be provided by the system. The results demonstrated that integration of the reasoning engine does not slow down the system significantly, even though additional communication is needed.

The qualitative evaluation explored the advantages and drawbacks of integrating the reasoning engine into Collect&Drop. This analysis is based on our own experience in implementing application logic with rule-based reasoning for

Collect&Drop. First, the reasoning component allowed distributing the functionality between application components. Second, we found it very convenient to implement, maintain and extend the application functionality with rules. Finally, the reasoning engine allowed us to implement a more intelligent system in terms of decision making, as users were released from choosing alternative resources by themselves. These achievements mainly came from the advantages of the rule-based reasoning approach outlined in [54], [55] and Section 2.3. Some of the disadvantages from Table 2 were met within the implementation. For instance, additional components were introduced to the system, hence this could increase the delay. Due to the small application scenario, the maintenance, inefficiency, and lack of reusability disadvantages were not observed. Also, learning to code with rules requires some time from developers.

A hard-coding approach works well for cases when the application requirements are known well in advance, and the application domain is static; that is, the domain does not require changes to application logic. Also, this approach works well if the target platform for application execution is a resource-constrained device. This way, the application developer can tailor the application to use resources efficiently. However, this approach has maintenance limitations.

To conclude, rule-based reasoning brings maintenance advantages to the system over a hard-coding approach. The main advantage we found is isolation of the rules from the compiled application. However, the exact benefits of the reasoning approach depend on the application domain, scenario, and requirements. For instance, not all the domains can be sufficiently described with rules. Also, hybrid approaches would work better if not all the concepts and their relations are known at the design time.

3.1.2 The scope of reasoning

The use of rule-based reasoning for a large-scale scenario is explored with a hypothetical student assistant scenario on a ubiquitous university campus, which is a smart space in the university area [34].

A general framework for such the scenario is presented in Figure 8. That is, context information is gathered by various resources, like sensors, services, and user profiles. This is a large-scale scenario, involving heterogeneous devices,

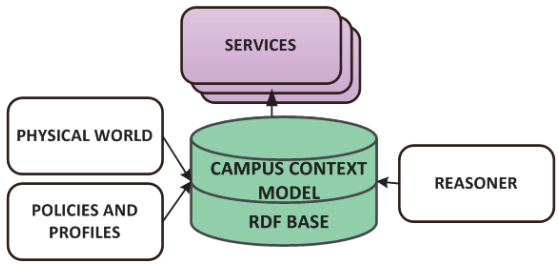

Fig. 8. Framework of context modelling and reasoning on a ubiquitous campus (Redrawn from [34]).

services, and users, and hence has heavy demands on common vocabulary for information sharing. An ontology-based model was selected to formally represent and organize the context information. A context reasoner performs inference to adapt the system behaviour and provide services to users.

As can be seen from Figure 8, the context model is a central component in the framework shared between other system components. We found the layered approach useful, where the upper-level context model describes the basic concepts related to context entities, while the lower-level ontology is focused on the campus domain. This approach supports reusability, as the upper-level context model can be shared across application domains. A similar layered design has been used in related work [109], [47]. The main concepts in the upper-level ontology are: the physical environment (describes the properties of physical surroundings), the digital environment (describes the digital devices, networks and services), and the social environment (describes people and their relationships). Concepts of lower level ontology are based on upper-level ontology. For instance, concepts of the social environment in upper-level ontology include Person, Profile, Social activity, and Social service. In the low-level ontology describing the campus domain, Professor and Student classes are subclasses of Person, Lecture is subclass of Social service, Meeting is subclass of Social activity, and Study plan is subclass of Profile. The context model was implemented with OWL.

The ontological approach for context model design allowed us to use hybrid reasoning that combines description logic based reasoning and rule-based reasoning. In our system, description logic based reasoning supports inference on concepts and relations between entities, while rule-based reasoning implements adaptive functionality. Some rule examples are presented in Table 4.

When the services and resources of the ubiquitous campus update ontology

Table 4. Modified versions of rules presented in [34] (in Jena syntax).

Reasoning rules

Rule1. Start ringing in the morning if the student is at home and according to her calendar she has a lecture today

```
[rule1:(?st rdf:type sw:Student)(?st sw:hasDevice ?ph)
 (?st sw:hasLocation sw:home)(?ph sw:hasAssistantService ?assist)
 (?assist sw:hasStatus 'morning_routine'^^xsd:string)
 (?assist sw:hasRingService ?ring)(?assist sw:hasCalendarService ?calendar)
 (?calendar sw:hasLecture ?lecture)(?lecture sw:hasStartTime ?lecture_time)
 (?assist sw:hasProfile ?profile)(?profile sw:hasWakeTime ?wake_before)
 currentTime(?current_time)timeDifference(?lecture_time,?current_time,?diff)
 equal(?diff,?wake_before)(?ring sw:hasStatus 'monitoring'^^xsd:string)
->
 drop(14)(?ring sw:hasStatus 'ringing'^^xsd:string)]
```

Rule 2. When the ringing is turned off, the weather and bus are checked and mail starts synchronization

```
[rule2:(?st rdf:type sw:Student)(?st sw:hasDevice ?ph)
 (?st sw:hasLocation sw:home)(?ph sw:hasAssistantService ?assist)
 (?assist sw:hasStatus 'morning_routine'^^xsd:string)
 (?assist sw:hasRingService ?ring)(?ring sw:hasStatus 'off'^^xsd:string)
 (?assist sw:hasWeatherService ?weather)
 (?assist sw:hasBusService ?bus)(?assist sw:hasMailService ?mail)
 (?weather sw:currentTemperature ?temp)(?weather sw:currentHumidity ?hum)
 (?bus sw:nearestBus ?nbus)(?mail sw:hasStatus 'unknown'^^xsd:string)
->
 drop(13)(?mail sw:hasStatus 'to_sync'^^xsd:string)
 (?assist sw:currentTemperature ?temp)
 (?assist sw:currentHumidity ?hum)(?assist sw:nearestBus ?nbus)]
```

Rule 3. Remind the student about the books which are soon to be returned to the library

```
[rule3:(?st rdf:type sw:Student)(?st sw:hasDevice ?ph)
 (?st sw:hasLocation sw:home)(?ph sw:hasAssistantService ?assist)
 (?assist sw:hasStatus 'morning_routine'^^xsd:string)
 (?assist sw:hasRingService ?ring)(?ring sw:hasStatus 'off'^^xsd:string)
 (?assist sw:hasMailService ?mail)(?mail sw:hasStatus 'sync'^^xsd:string)
 (?assist sw:hasLibraryService ?library)
 (?library sw:hasBooksToExpire 'true'^^xsd:boolean)
 (?library sw:hasBookList ?link)(?assist sw:hasReminder ?reminder)
->
  drop(4)(?assist sw:hasStatus 'day_routine'^^xsd:string)
  (?reminder sw:hasLink ?link)]
```

Rule 4. Switch the mobile phone of a student to the calm mode if she is in a lecture.

```
[rule4: (?st rdf:type sw:Student)(?st sw:hasDevice ?ph)
 (?ph sw:hasMode 'general'^^xsd:string)(?ph sw:hasAssistantService ?assist)
 (?st sw:hasLocation ?location)(?assist sw:hasCalendarService ?calendar)
 (?calendar sw:hasLecture ?lecture)(?lecture sw:hasLocation ?location)
 (?lecture sw:hasStatus 'active'^^xsd:string)
  ->
  drop(2)(?phone sw:hasMode 'calm'^^xsd:string)]
```

Rule 5. Remind the student about lunch, with the menu, at the time specified in the user profile

```
[rule5:(?st rdf:type sw:Student)(?st sw:hasDevice ?ph)
 (?ph sw:hasAssistantService ?assist)
 (?assist sw:hasStatus 'day_routine'^^xsd:string)
 (?assist sw:hasProfile ?profile)(?assist sw:hasRestaurantService ?rest)
 (?rest sw:hasStatus 'active'^^xsd:string)
 (?profile sw:hasLunchTime ?lunch_time)
 currentTime(?current_time) equal(?lunch_time,?current_time)
 (?rest sw:hasMenu ?menu)(?assist sw:hasReminder ?reminder)
->
 drop(6)(?rest sw:hasStatus 'done'^^xsd:string)(?reminder sw:hasLink ?menu)]
```

with actual data, reasoners perform reasoning on the updated information and command which services should be executed and what data should be utilized by them. Reasoning results may be reflected in the ontology as well.

The results from this study are the following: First, a large-scale ubiquitous scenario requires careful context modelling. That is, in addition to general requirements on context modelling [7], [46], the following are considered important: 1) A context model for large-scale scenarios should allow easy integration of new domains and services. Moreover, maintenance of a large context model can be challenging. For such cases, the layered approach can be useful. 2) Large-scale scenarios may involve devices which cannot afford a certain context modelling approach due to limited resources. Also, certain modelling approaches might not be suitable for highly dynamic contexts, like sensor measurements. For such cases, a hybrid approach to model the context can be proposed [136], [137]. Another option is to design lightweight knowledge transfer protocols [138].

Second, large-scale scenarios may involve large number of resources and services producing a large amount of information. Hence, certain design and technological solutions could be required to provide sufficient quality of service. For example, reasoning over a centralized context ontology containing highly dynamic data might be time consuming. Distributing the context model and reasoning may overcome these issues [8],[139],[76]. Moreover, distribution may facilitate privacy support, assuring that sensitive data are not sent to other components [8].

The QuizBlasters system, presented in Publication III, was implemented with these considerations in mind. QuizBlasters is a learning application which combines elements of a treasure hunt and a multiplayer action game. This application has two modes: a learning mode and a multi-player game mode. In the learning mode, a user explores the smart space with his mobile phone and answers quizzes by touching corresponding NFC tags placed in the environment. Different bonuses, which can be later used in the game mode, are given to the user, based on his progress. In the game mode, users challenge each other by playing the game on a wall display. In this mode, mobile devices serve as remote controllers for the game. More details on entering the game mode will be given in Section 3.2.1.

QuizBlasters' architecture is presented in Figure 9. In this system, rule-based reasoning supports the decision-making process utilizing various contexts. The reasoner consists of a reasoning server and multiple reasoning clients deployed into

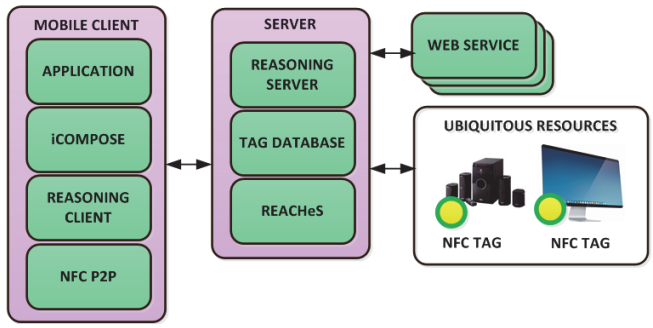

Fig. 9. Architecture of QuizBlasters, modified from Publication III.

mobile clients. This way, a distributed reasoning mechanism is utilized, which implies that the reasoning can take place on the mobile device (’local reasoning’) or on the server (’remote reasoning’).

Reasoners operate on corresponding local and remote context models, having partially intersected sets of concepts. This design solution allows the preservation of privacy, so only the concepts shared by both local and remote context models are transferred over the network. The reasoning client is implemented in J2ME and deployed on mobile phones as a part of the mobile client. The reasoning client uses the m-Prolog inference engine optimized for J2ME applications [140]. Table 5 demonstrates some rules.

The following results were obtained from this study. The distribution of context modelling and reasoning to local and remote parts provided certain benefits. First, this approach allows us to address users’ privacy preferences, because user sensitive information can be processed locally on mobile phones. Second, this approach can reduce any unnecessary communication overhead, as information required for reasoning is available on the mobile phones and there is no need to send it or request additional information over the network. This approach is easy to implement, and it serves the purposes of the QuizBlasters application well. However, all the components must share the common context model concepts. Some approaches have been suggested to overcome this issue [73],[141].

Experiments with resource limited devices, like mobile phones, revealed the following: First, for memory economy, the size of the local knowledge base should be limited. For instance, the knowledge base could be cleaned periodically, so that only the most relevant facts are kept. Second, knowledge base consistency should be supported due to limited battery life, for instance by making backups. To address these issues holistic approaches have been suggested, developing context

Table 5. Example of QuizBlasters' rules.

Situation description and examples of the rules

Situation 1: A user wants to join the game which is played on a wall display. This is possible only when the user has no scheduled classes at the moment and other players are from his class.

```
result(join_game):-application_activity(idle),display(reserved),
                   application(game),calendar_activity(idle),
                   sameclass(true).
```

Situation 2: If user is from different class, the corresponding message will be shown.

```
result(wrong_game_class):-application_activity(idle),
                          display(reserved),application(game),
                          calendar_activity(idle),sameclass(false).
```

management frameworks specifically for resource constrained devices, both involving a manual coding approach [114] and knowledge-based techniques [142].

As a conclusion, rule-based reasoning has a wide scope in implementing smart space applications. Both large-scale scenarios and applications involving resource-constrained devices may benefit from using rule-based reasoning. The developer has to consider carefully the capabilities and limitations of concrete smart space and how they match the requirements of a concrete ubiquitous system.

3.1.3 Supporting smart space interaction with reasoning

Smart spaces would be meaningless without humans who interact with each other and the smart space. This also raises questions on how smart spaces support social interaction like the ones presented in Publication V: "Can smart spaces take part in social interaction and support it? Can smart spaces become participants in social interaction and to what extent?" These are challenging and important issues, especially within the Internet of People paradigm [143]. This thesis does not attempt to answer these questions. Instead, we analysed our own experience in creating ubiquitous applications, and related their reasoning component architecture to interaction support. We defined smart space interaction as "*the sequence of bidirectional interactions between actors (or a group of actors) which take into account context, including the actions and behaviour of these actors.*" Actors of such interaction are "*independent entities, able to sense and receive information, act on their own, and communicate with each other through the environment.*" Moreover, actors can initiate interaction with others, i.e. active interaction. In addition to humans, an object of a smart space can become an actor, if it possesses the properties outlined above. Publication V suggests using

knowledge based technologies to enrich objects of a smart space with capabilities of capturing, interpreting and reacting in situation as they offer a) means to operate with different objects of the system with unified semantics; b) solutions to define actions to take according to situation; c) efficient algorithms for decision making. Moreover, we explore how reasoning distribution supports smart space interaction.

Publication V defines four types of interaction in a smart space: User-to-User, User-to-Resource, Resource-to-Resource, and Facilitated interaction. User-to-User is the form of smart space interaction where the interacting actors are humans. In User-to-Resource interaction, one actor is human, and another actor is a resource of the smart space, i.e. a physical artefact of the environment. In Resource-to-Resource interaction both actors are resources. Facilitated interaction is a specific form of interaction combining all the other types, and refers to active interaction of smart space resources to achieve some utility for users, like to initiate interaction. To see how the proposed types of smart space interaction are supported by different reasoner architecture, we analysed three ubiquitous systems developed by us: EMA, QuizBlasters, and SI.

EMA is a context-aware reminder for mobile phone users [10]. This application uses location, points of interest, and user tasks information to generate appropriate reminders for the users. The QuizBlasters application was already discussed in Section 3.1.2. SI is a mobile application allowing the user to find people with similar interests. Users specify their hobbies and the types of alerts they expect to receive, also create the rules to define which hobbies they are interested in. This application uses Bluetooth to notify about other users around having similar interests. All our prototypes treat smart space as an active participant in the interaction. For instance, QuizBlasters motivates users to interact with the infrastructure and the resources of a smart space. EMA and SI constantly perform decision-making to provide social utility for users. All the applications implement rule-based reasoning for their decision making.

We implemented these prototypes gradually with more and more distributed context reasoning, as can be seen from the Figure 10. The EMA system presents a centralized approach (Figure 10a). Mobile clients contain just GUIs. The server hosts an OWL ontology knowledge base and a rule-based reasoner. The QuizBlasters application (Figure 10b) utilizes a hybrid approach. Mobile clients host their own reasoners and context models, and also server reasoning is required to support interaction between multiple mobile devices.

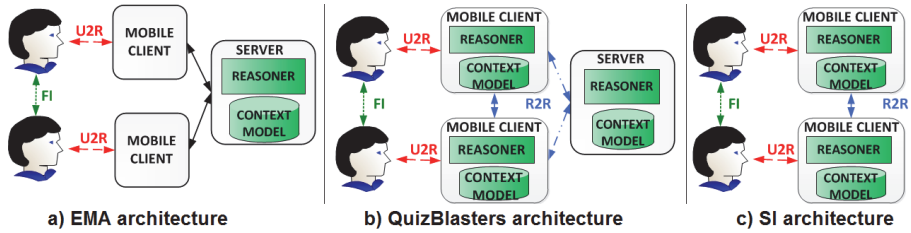

Fig. 10. Comparison of prototype architectures (FI-Facilitated interaction, U2R-User-to-Resource interaction, R2R-Resource-to-Resource interaction).

The SI system (Figure 10c) demonstrates a distributed approach. Mobile clients host both context models and reasoners for decision-making, and no server is needed. The knowledge base of SI is separated into private (user sensitive facts and rules) and public profiles (shared facts and rules). The public profile has a context model predefined for all the clients. The private profile can be unique for each client. Rule-based reasoners of the mobile clients use rules from both public and private profiles for decision-making.

The main results from this study are the following: The architecture of the system, with respect to the reasoning components does not have a big effect on implementing the four types of smart space interaction. This means that a well-designed architecture, whether it is a centralized, hybrid, or distributed, may serve well for all four smart space interaction types. However, for a centralized approach, Resource-to-Resource interaction is often replaced with plain M2M communication, where the actors are not equipped with capabilities to reason about the situation context. The system architecture has an effect on the flexibility and intelligence of the smart space interaction. For instance, if the component embeds its own reasoning, it can customize its own decision-making. Also, distributed reasoning provides flexible mechanisms to support functionalities which otherwise require specific methods. For example, our SI system naturally supports privacy as no user sensitive information is sent over the network. For a centralized approach, a specific mechanism should be implemented to guarantee that user sensitive information is not exposed, like anonymization [144]. However, different factors need to be considered when designing the application architecture, like smart space resources and application requirements [145]. Table 6 summarizes the observed advantages and drawbacks of realized architectures.

Table 6. Prototypes implemented for Publication V.

Item	EMA	QuizBlasters	SI
Knowledge based architecture	Centralized	Hybrid approach	Decentralized
Advantages	- Easier control and development for social communities (e.g. sharing tasks) - Thin mobile client	- Privacy support - Efficiency in terms of response time - Still easy control and maintenance with server-side reasoning	- Privacy support - Efficiency - Decreased network overload
Disadvantages	- Constant network connectivity - Limited scalability - Limited privacy	- Constant network connectivity for cases involving user interactions - Thick client	- Thick client - Difficult implementation - Users need to know how to create profiles and rules and keep them consistent - Thick client

3.2 Research on reasoning to create applications

The second research question of this thesis is related to understanding how rule-based reasoning can be used to create ubiquitous applications. The QuizBlasters prototype described in Publication III contributes to understanding on how the rule-based reasoning can be used to enable and control the application composition process with physical user interfaces. Perception Framework described in Publication II explores creating ubiquitous services with middleware support.

3.2.1 Interactive application composition

The QuizBlasters application was already described in Sections 3.1.2 and 3.1.3. This section focuses on the application composition functionality of QuizBlasters and how rule-based reasoning supports it.

QuizBlasters can be referred as a system supporting interactive application composition [79]. This means that the functionality of the application can be composed from several application primitives or other services by a user directly interacting with the system and the environment. Moreover, QuizBlasters uses a touch-based physical user interface in addition to graphical user interfaces. This interface is based on NFC technology and facilitates users' interaction with objects

of the smart environment and with others with mobile phones that are equipped with NFC readers.

QuizBlasters uses an application composition approach in the multiplayer action game mode. In this game, the users have to hit each other's avatars using various weapons. The users control their avatars and the game process on a wall display using their mobile devices as remote controllers. Here, the game is an application to compose. Users need to find a display to play the game, and at least two players are needed. These requirements are met by following a direct manipulation principle facilitated by an NFC-based physical browsing interface. To select a display, users have to touch the corresponding tag with their phone. To select a player, users have to touch the phone of the other person with their phone (a so called handshake operation).

Rule-based reasoning supports three main functions in the game mode: a) it is responsible for adaptive behaviour, b) it supports the game composition process, c) it specifies the 'rules of engagement', which dictate when, where and with whom the players can compose and play the game. This functionality is implemented with a distributed rule-based reasoning mechanism (Figure 10). Reasoning clients serve for local context-awareness and adaptation tasks, and they are deployed into each mobile client. The reasoning server handles the game composition process. Both reasoning clients and the server participate in the reasoning of game engagement.

Whenever a user performs an action, for example, touches a tag or the phone of another user, a reasoning client processes it further. The reasoning client may request additional context information in order to process a specific event, for example, the availability status of a display when a user attempts to compose a game using a particular display. All this context information updates the local knowledge base. When all the required information about an event is collected, a decision is made about the action to be taken. This information is either sent to other mobile client application components (triggering context-aware adaptation), or to a reasoning server (composition).

The reasoning server infers decisions when multiple interacting users attempt to initiate application composition, for example, when new players try to start playing a game. In this case, a reasoning client sends a request to the reasoning server. The reasoning server handles the event, infers a decision, and delivers it to all the users involved in this event. Then reasoning clients notify the corresponding application components about the decision. Game composition requires certain conditions to be met, such as at least two players and an idle display. Moreover, additional restrictions can be considered as rules of engagement, like only players

from the same school class can play together. Table 7 demonstrates some examples of the rules.

Table 7. Rules example for game composition process.

Situation description and examples of the rules

Situation 1: Users made a handshake and the local reasoner decides based on the situation if the request for game composition can be sent to a remote reasoner.

```
result_on_handshake(send_request):-application_activity(idle),
                                   calendar_activity(idle).
result_on_handshake(send_request):-application_activity(game),
                                   calendar_activity(idle).
result_on_handshake(quiz_abandon):-application_activity(quiz),
                                   calendar_activity(idle).
```

Situation 2: The remote reasoner checks classes and the state of the display to allow the playing of the game. Also, in case there is already game on some other display with the same class, users will be suggested to join it alternatively.

```
result(no_display_game,User1,User2,Display,Game,AlternDisplay):-
                    same_class(User1,User2,true),
                    display(Display,busy),
                    matching_game(User1,Game,AlternDisplay,false).
result(join_game,User1,User2,Display,Game,AlternDisplay):-
                    same_class(User1,User2,true),
                    display(Display,busy),
                    matching_game(User1,Game,Display,true).
result(join_alt_display,User1,User2,Display,Game,AlternDisplay):-
                    same_class(User1,User2,true),
                    display(Display,busy),
                    matching_game(User1,Game,Display,false),
                    matching_game(User1,Game,AlternDisplay,true).
result(create_game,User1,User2,Display,Game,AlternDisplay):-
                    same_class(User1,User2,true),
                    display(Display,idle),
                    matching_game(User1,Game,AlternDisplay,false).
result(create_or_join,User1,User2,Display,Game,AlternDisplay):-
                    same_class(User1,User2,true),
                    display(Display,idle),
                    matching_game(User1,Game,AlternDisplay,true).
```

The following results are obtained from this study: We found rule-based reasoning together with a physical browsing interface convenient for composing a multiuser game in QuizBlasters. First, rules allow formulating the logic of composing the game in a flexible and modifiable manner, supporting the advantages itemized in Table 2. Second, rules are convenient for implementing the 'rules of engagement' describing when and how the game could be composed, therefore a context-aware application composition was achieved [87],[89]. Also, due to familiarity with the rule metaphor, it was easy to explain to users the game composition concept involving concrete resources of smart space and other users. Users feel they are in control when they control the application, understand how the application makes decisions and expect the correct actions to follow [146],[147]. Each action of the user of QuizBlasters and the corresponding messages from the reasoners were

demonstrated on a specifically designed GUI, supporting users in the composition process. By direct manipulation with resources and users of a smart space, users of QuizBlasters initiated system actions and could predict the system response by themselves. Hence, such a design solution facilitated a feeling of being in control in QuizBlasters [147], [148]. From the development point of view, this approach has all the advantages listed in the Section 3.1. However, the implementation required additional efforts in order to make the system robust in the case of device or network failures, like the use of timeouts to coordinate handshakes. If the handshake request from one device for some reason did not reach the server, all the corresponding devices were released and a corresponding message was shown to users. Overall, we find rule-based interactive application composition for such scenarios to be flexible.

3.2.2 Creating services with middleware

Creating ubiquitous services with middleware support is explored in Publication II. This article proposes Perception Framework, a service-oriented middleware supporting the development of ubiquitous services. The focus of this middleware is above the low-level issues, like hardware.

Perception Framework allows the construction of service logic with rules and the available context. Developers describe service logic with rules utilizing elementary contexts available in Perception Framework. The Perception Framework middleware on the fly resolves the required context dependencies and retrieves context data for service execution. This way the service is dynamically created from the available elementary contexts by locating the required components and configuring them to provide the required information. Moreover, Perception Framework uses Rule Interchange Format (RIF) [56] for writing rules, hence developers are not restricted with the rule language.

Functionally, Perception Framework is divided into three layers: the Sensing layer, the Semantic layer, and the Control layer. The Sensing layer provides mechanisms to retrieve sensor data. The Semantic layer is responsible for annotating data with semantics and inferring new knowledge. The Control layer offers maintenance functionality, like resolving conflicts. The components of the Perception Framework middleware are presented with Figure 11. More details

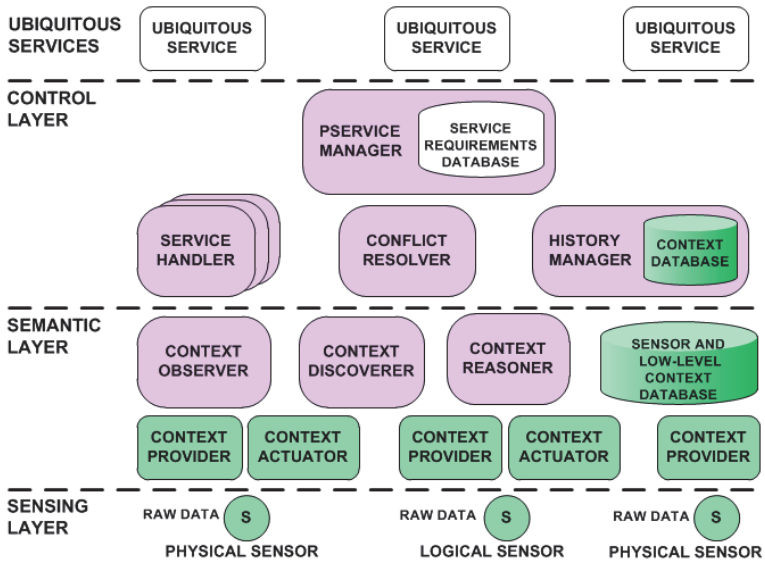

Fig. 11. The components of Perception Framework.

about each layer are given in Publication II. Here we concentrate on the rule-based reasoning support for the creation of a ubiquitous service.

This middleware operates with context, and allows developers to construct the high-level context and define the adaptive behaviour of the ubiquitous services by using rules and low-level contexts. Low-level context is information characterizing an entity, and this information is retrieved either directly from sensors, or from measurements pre-processing. Context providers (Figure 11) map the raw data of sensors to low-level contexts. These low-level contexts are structured with an ontological context model in Perception Framework. The context model is presented with five main concepts: sensor, context, actuator, parameter, and value. The sensor concept describes actual sensors, and it is used mostly for informative purposes. The context concept describes the information that a certain context provider produces. An actuator describes the behavioural changes associated with the context actuator. Parameters describe the necessary information required to obtain certain contexts. The value concept describes an actual context data supplied by the context provider. This low-level context ontology creates a common vocabulary to be used within Perception Framework.

Service developers use these low-level contexts to write the application logic of their ubiquitous services with rules. Service logic rules need to be written in RIF,

which is XML-based syntax, allowing rule transfer between systems using different rule languages. Perception Framework supports two RIF dialects, RIF production rule dialect (forward chaining) and RIF basic logic dialect (backward chaining). Perception Framework translates RIF rules into corresponding internal languages of the rule engines. Win-Prolog is used for backward chaining, and Jess is used for production rules. Also, Perception Framework defines the compulsory vocabulary that service developers must utilize in order to inform the middleware about some dependencies and events, like known facts and information about the context changes that are of interest to the service. This vocabulary is defined as n-tuples. For instance, the OBSERVER(context,condition,value) tuple tells that the service is interested in knowing when the value of the context satisfies the condition. For a full list of items for compulsory vocabulary, the reader is referred to Publication II. When the rules are ready, the developer registers the service in Perception Framework with the provided interface.

When the service is registered in Perception Framework, the context reasoner handles the service rules. It checks the rules and loads the corresponding rule engine. The reasoner loads all the known facts and rules into its working memory and starts inference. The reasoner queries other components of Perception Framework in case of unknown contexts, as well as to assign observers. When reasoning is finished, the corresponding message is delivered to the service. The reasoning cycle repeats when the context in which the service is interested changes.

To evaluate the Perception Framework middleware we implemented a transport assistant scenario in both ways: with Perception Framework and without it, i.e. from scratch. This service provides recommendations regarding means of transport based on user weather preferences and locations defined in calendar events. Examples of rules written for this scenario in RIF basic logic dialect syntax and corresponding translation to WIN-Prolog are demonstrated in Table 8.

Our comparison demonstrated that implementing a ubiquitous service with Perception Framework requires much less effort in terms of written code. Moreover, we found it quite convenient to express service logic with rules. Once implemented, contexts available in Perception Framework can be shared between several services. However, we have to admit, that this advantage comes with the price of processing time delays, caused by communication between Perception Framework components.

The following results were obtained from this study: as we discussed in Section 3.1, a rule-based approach is convenient to implement application logic.

Table 8. RIF and corresponding Win-Prolog rules for transport assistant service.

Simplified rules for transport assistant service (slightly modified from Publication II)

RIF rules:

```
Document(
  Prefix(xs <http://www.w3.org/2001/XMLSchema#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(cnt <http://example.com/contexts#>)
  Prefix(cpt <http://example.com/concepts#>)
  Group
  ( Forall ?Distance ?Temperature ?Humidity (
      cpt:foot(?Distance ?Temperature ?Humidity) :-
        And(External(pred:numeric-less-than-or-equal(?Distance 2))
            External(pred:numeric-greater-than-or-equal(?Temperature 10))
            External(pred:numeric-less-than-or-equal(?Humidity 50))))
    Forall ?Distance ?Temperature ?Humidity (
        cpt:bus(?Distance ?Temperature ?Humidity) :-
          Or(External(pred:numeric-greater-than(?Distance 2))
             External(pred:numeric-less-than(?Temperature 10))
             External(pred:numeric-greater-than(?Humidity 50))))
    Forall ?Distance ?Temperature ?Humidity (
      cnt:main :-And(cnt:fact(''MapContext.Distance'' ?Distance)
                     cnt:fact(''WeatherContext.Temperature'' ?Temperature)
                     cnt:fact(''WeatherContext.Humidity'' ?Humidity)
                     And(Or(And(cpt:bus(?Distance ?Temperature ?Humidity)
                                cnt:result(''BusContext.Route''))
                            And(cpt:foot(?Distance ?Temperature ?Humidity)
                                cnt:result(''Go by foot'')))))))
cnt:fact(''CalendarContext.CalendarName'', ''my_calendar'')
cnt:parameter(''BusContext.DepartureStreetLocation'',
              ''CalendarContext.CurrentEvent.EventLocation'')
cnt:parameter(''BusContext.DestinationStreetLocation'',
              ''CalendarContext.NextEvent.EventLocation'')
cnt:parameter(''BusContext.DateParameter'',
              ''CalendarContext.NextEvent.StartTime'')
cnt:parameter(''MapContext.DepartureStreetLocation'',
              ''CalendarContext.CurrentEvent.EventLocation'')
cnt:parameter(''MapContext.DestinationStreetLocation'',
              ''CalendarContext.NextEvent.EventLocation'')
cnt:observer(''CalendarContext'')))
```

Corresponding Win-Prolog rules:

```
fact(`CalendarContext.CalendarName`,`my_calendar`).
parameter(`BusContext.DepartureStreetLocation`,
          `CalendarContext.CurrentEvent.EventLocation`).
parameter(`BusContext.DestinationStreetLocation`,
          `CalendarContext.NextEvent.EventLocation`).
parameter(`BusContext.DateParameter`,
          `CalendarContext.NextEvent.StartTime`).
parameter(`MapContext.DepartureStreetLocation`,
          `CalendarContext.CurrentEvent.EventLocation`).
parameter(`MapContext.DestinationStreetLocation`,
          `CalendarContext.NextEvent.EventLocation`).
observer(`CalendarContext`).
foot(Distance, Temperature, Humidity):-compare(=<,Distance,2),
              compare(>=,Temperature ,10),compare(=<,Humidity,50).
bus(Distance,Temperature,Humidity):-  compare(>, Distance,2);
              compare(<, Temperature,10); compare(>, Humidity,50).
main:- fact(`MapContext.Distance`,Distance),
       fact(`WeatherContext.Temperature`,Temperature),
       fact(`WeatherContext.Humidity`,Humidity),
     (foot(Distance,Temperature,Humidity),result(`BusContext.Route`);
        bus(Distance,Temperature,Humidity),result(`Go by foot`)).
```

This study also supported this observation.

Perception Framework supplies a set of interconnected components facilitating development and execution of ubiquitous services. In this sense, it is similar to other solutions proposed for the development of context-aware applications [102],[104],[109],[110]. Perception Framework provides a common vocabulary for all the physical and logical (Web services) sensors. Hence, internal communication of participating services and sensors is hidden from ubiquitous service developers; only the information that matters is exposed, like the context. Perception Framework provides elements handling service execution and maintenance as some related work [38],[103]. Developers can build the application logic of ubiquitous services by using rules and available context information, as seen in some proposals [109],[38]. Perception Framework handles the overall execution.

Perception Framework provides specific vocabulary to define dependencies between actual context instances in the ubiquitous application, such as the parameters to retrieve the context. This way, developers are flexible in constructing relationships between context instances, but they are responsible for ensuring that such dependencies are valid. This vocabulary is used when ubiquitous service rules are designed. Hence, it can be easily modified without ubiquitous service recompilation. Another approach would be to implement application domain context models, determining such dependencies explicitly [109], [38]. This approach is close to other service composition solutions, like OWL-S [83].

RIF support in Perception Framework provides a general way of writing application logic rules, without the need to learn the proprietary rule format as in related work [38],[110]. This is also a step promoting the use of standardized techniques in the ubiquitous computing domain.

Implementing the ubiquitous service with Perception Framework can reduce developer's efforts. On the other hand, Perception Framework, as with other similar proposals [108], [109], introduces communication links, hence delay could increase. However, it is difficult to estimate actual increase in delay as the prototype systems developed in Publication II were not optimised for performance. Another open issue of using such middleware is possible limitations to on the fly modifications of the executing services. Perception Framework requires services to register themselves before these services are executed. Hence, if modification in the rules is needed, then the corresponding service should be unregistered first.

Overall, a rule-based reasoning approach to create the logic of a ubiquitous service is convenient. Additional effort is required from developers to instruct Perception Framework with specific concepts used in rules on how to handle

context dependencies and events. Finally, the use of such a framework is justified when developed context providers are utilised by several ubiquitous services.

3.3 Research on self-adaptation in smart spaces

The third research question of this thesis is related to the self-introspection of smart spaces. To approach it, first a general architecture is proposed, which is then verified with simulation and a real-world prototype.

A general framework to support meta-level control in smart spaces is presented with Publication IV. The same article proposes a simulated smart space scenario following this framework. Publication VI demonstrates how ubiquitous learning applications can be equipped with meta-level control. A real-world prototype implementing the presented framework is described in Publication VII with the Driving coach system. This system supports drivers in their fuel efficient driving.

3.3.1 Conceptual framework

Smart spaces are highly dynamic interactive environments equipped with technology to adapt themselves and provide context-aware applications that support users in their daily lives. Hence, smart spaces require management efforts to provide valuable support to their users. As we observed in Section 2.6, autonomic computing tackles such challenges. In other words, autonomic computing facilitates self-configuration, self-optimization, self-healing, and self-protection properties. This thesis studies the self-adaptation of smart spaces and their applications. A meta-level control framework to achieve self-adaptation is proposed.

The meta-level control framework distinguishes context-based adaptation activities of smart spaces from monitoring and controlling these adaptation activities. This is achieved by placing the reasoning about adaptation activities to a separate level, called Meta-level. This approach produces clear system design, customizability, and easier reuse [149]. Clear design is achieved by assigning different types of tasks to separate layers. Hence, improvements of modularization and system maintenance can be achieved. Customization is achieved by tailoring control decisions to specific situations. Reusability is achieved by defining the interfaces so that several applications can reuse the same monitoring and control functionality. Generally, such system design results in feedback loop system properties. Namely, the system modifies itself based on its real use, performance

criteria, and user and environment context. This framework is inspired by Cox and Raja's research [32],[150],[151].

Such a meta-level concept has been earlier applied to different research areas, like research on agents [151],[152] and machine learning [153],[154]. This thesis applies it to smart spaces and their corresponding applications.

Traditional ubiquitous applications can be represented as an Action-Perception loop (Figure 12). A ubiquitous system senses the user and the environment (Ground level), and reasons the actions to be performed in response to changes in the user behaviour and environment (Object level). These actions, in their turn, can change the environment and affect user behaviour. This may trigger new system actions, so the cycle continues. Meta-level takes the control of this cycle (Meta-level). That is, it adds self-introspective monitoring and control over the reasoning process. This metareasoning [32],[150] (Meta-level) evaluates reasoning actions and modifies them as necessary. Hence, Meta-level aims to improve the quality of decision-making. In ubiquitous systems, user satisfaction is an important criterion for evaluating the decision-making of the system [155]. Hence, we include a monitoring and feedback link connecting Meta-level and Ground-level (Figure 12). This link allows monitoring how well reasoning tasks support users, as well as providing an explanation support for users.

The suggested framework is presented with Figure 13. We form a Ground level with components sensing the environment and the user (Perceptors), as well as components making actions and delivering information to the user (Actuators). Object level components implement the actual functionality of the applications. For example, in the case of service-oriented architecture of applications, these

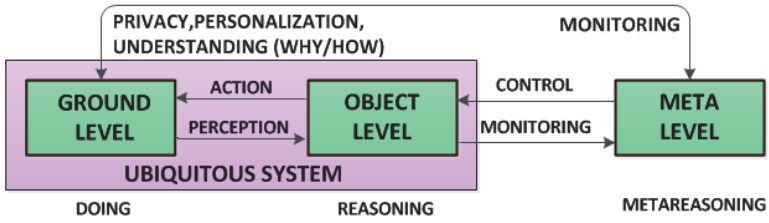

Fig. 12. Meta-level concept, modified from [32].

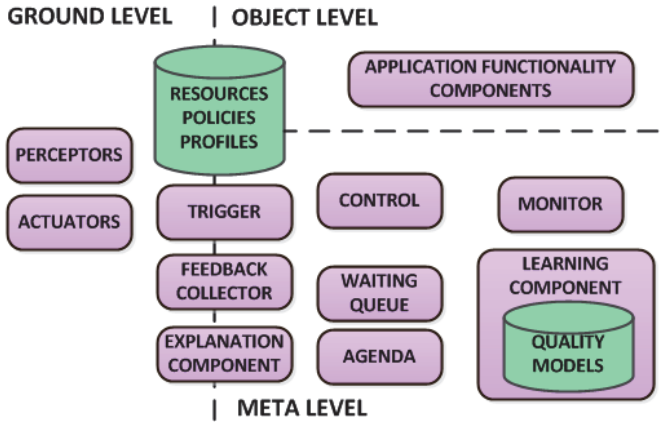

Fig. 13. Meta-level framework.

components could be a Service composer, a Service deployer, and a Service executor. Meta-level equips the system with monitoring and control functionality. A Trigger component listens for Ground level events and sends them to the Control component that is responsible for controlling object level tasks. The Control component also decides which tasks should be processed immediately, and which could be shortly postponed (Agenda). A Waiting queue gathers events which cannot be processed at the moment because of the lack of supporting context information (e.g., the requested device is not available at the moment). To control object-level tasks, the Control component consults Quality models that contain the best configurations of object-level tasks strategies and system performance. The Monitor and Learning components are responsible for Quality models construction. At the same time, the Control component informs the Explanation component about decisions made. The Monitor component monitors object-level task execution and feeds the Learning component with this information. The Resource repository, Policies, and Profiles contribute to all levels. Feedback collector can be considered as a Perceptor, but for convenience, we treat it separately.

This framework provides a general architectural solution to implement meta-level support for smart space applications. The concrete implementations of the components differ depending on the specifics of the targeted applications. Such an approach allows exploring the use of the framework in different application scenarios, for instance Publication VI applies it to a ubiquitous learning system. The proposed framework resembles the ideas of the MAPE-K loop reference model [118], where the management tasks are layered separately. With such a framework the smart space can implement both introspection (ability to observe its own

behaviour), and intercession (ability to act on these observations to modify own behaviour) activities [149]. Hence, the smart space can achieve self-adaptation.

The solution proposed in this thesis, as with some related work [121], [120], [123], is heavily based on monitoring and control. However, in addition to specifying which actions to perform in concrete situations [121], in the proposed solution, learning can be used to perform control actions as well. The meta-level control framework may require implementing different solutions for object level tasks, similarly to the application construction from the units in [124],[125],[126], so that the meta level would be able to substitute the solution with a more appropriate one. That is, the meta-level control framework alters object-level tasks. In fact, the proposed framework does not limit object and meta level implementation to a certain technique. Different approaches, like rules [121],[123] or appropriate algorithms [120] can be used for both object and meta levels. However, only a centralized approach is considered in the proposed Meta-level framework. That is, a single meta level implements management tasks for the smart space. This provides the advantages of a holistic view on the performance and personalisation of the smart space applications, however it presents certain challenges, such as dependency on interfaces. It is an open issue how distributed architecture can be realized to support self-adaptation in a smart space with the Meta-level framework [156],[32].

3.3.2 Self-adaptation prototypes

Publication IV presents the first experiment on the meta-level control framework. This experiment is a simulation study delivering appropriate services to users according to their location, preferences, and feedback. This study demonstrates the division of functionality of smart space to object and meta levels. The simulation is implemented with Win-Prolog and demonstrates the use of rule-based reasoning to implement both object level and meta level functionality. The simulated smart space consists of several rooms, equipped with resources able to execute services. Services can be executed on several resources, each providing a certain level of quality. User preferences include quality requirements for service execution.

The main concepts of the context model are presented with Table 9. These concepts describe the resources and services that the smart space offers (service_profile, device_profile), as well as the user preferences for the services and locations (user_profile). This information is used by the Object layer to deploy and execute services on the proper resources. Performance and satisfaction

information is used by the meta-level for decision-making. For instance, the information about the quality of the deploying mechanism used helps the meta-level to estimate the performance of the system for a certain deployment task (best_quality_model). User satisfaction reflects user opinion about system performance (user_satisfaction).

In this study, the object level is presented with service deployment and execution tasks (see Section 3.3.1). The service deployment task is presented with four possible strategies which consider the criteria of privacy, quality, and availability. The privacy and quality criteria present user preferences regarding public use and required capabilities of resources for certain services. Availability tells if the resource is currently in use. Deployment strategies consider only the cases where at least two requirements are satisfied. If one or no requirements are satisfied, then the user is asked whether the service should be put into the Waiting queue. The Service execution task just executes the selected deployment strategy.

Table 9. Context model of simulated smart space (modified from Publication IV).

Predicate	Description
`user_profile(user_name, location, service_id,action, privacy, priority, QoS, time, mode)`	Describes user preferences about the services and locations. Also describes preferences regarding privacy, quality, priority, accepted start time delay and if the service should be started automatically or manually (mode).
`service_profile(service_id, interface, format)`	Describes functionality and supported formats of the service.
`device_profile(device_id, interface,format,quality, publicity, latency)`	Describes functionality, data format, guaranteed quality, and publicity of the device. Also latency (ms) is described telling how much time it takes to start executing the certain function (ms).
`best_quality_model(service_id, location, pqa_depl_time, pnotqa_time, notpqa_time, pqnota_time)`	Describes best deployment time for different strategies.
`user_satisfaction(user_name, service_id,action,location, device_id,desired_quality, achieved_quality,pqa_time, pnotqa_time,notpqa_time, pqnota_time, user_satisfaction_grade)`	Describes user satisfaction with service deployment on the specific device. Also, corresponding deployment latency is registered.

Object level tasks are implemented with Win-Prolog rules. These rules have time cost, so that it is possible to measure the performance of the rules in terms of time. Table 10 shows an example of a rule locating the service to available resources satisfying privacy and quality requirements.

The meta-level task monitors and controls the execution of object-level tasks. Moreover, as was discussed in Section 3.3.1, the link to ground level is required to deliver a certain quality of service. Examples of meta-level tasks in the simulation are: deployment strategies prioritizing, controlling which services should be deployed immediately, monitoring service execution performance and commanding redeployment if the quality of service is not sufficient. Meta-level tasks are also implemented with Win-Prolog rules, the example is in Table 11.

The simulated smart space handled the events of new users entering the smart space, the appearance or failure of resources, and the starting and stopping of new services. Also, the simulated smart space gathered user feedback regarding service allocation and execution tasks.

Publication VI explores user support in ubiquitous learning systems. Four main user roles are identified: learner, instructor, developer, and researcher. The learner uses the learning system to learn a specific subject. The instructor creates and controls the learning activities and content. The developer implements the learning system. The researcher analyses the system based on predefined criteria. Publication VI analyses how all these four types of users are supported in the

Table 10. Object-level rule example in the simulation study.

Description and rule example
Tries to locate the service to the available resource which satisfies privacy and quality requirements set by the user.

```
if_pqa(User,Location,Service,Resource,Privacy2,QoS,WastedTime,`no`, Cost):-
    time(1,I),I=(_,Start),
    user_profile(User,Location,Service,Action,Privacy,Priority,QoS,
                                                    Time,Mode),
    service_profile(Service,Action,Format),
    device_profile(Resource,Action,Format,QoS,Privacy2,AllocationTime),
    device_location(Resource,Location),
    (device_state(Resource,idle)->
          (compare(=,Privacy,Privacy2);compare(=,Privacy,0))
            ->
             (GivenTime is Time-WastedTime,
              compare(>=,GivenTime,AllocationTime)->
    if_pqa(User,Location,Service,Resource,Privacy2,QoS,WastedTime,
                                                    `yes`,Cost)
              )
      ),
    time(1,U),U=(_,End),
    Cost is End - Start, !.
```

Table 11. Meta-level rule example in the simulation study.

Description and rule example

Defines if the service request should be processed now can it be placed to Agenda (see Section 3.3.1). This decision is defined with priority and the allowed time delay (see Publication IV).

```
put_task_to_agenda(User,Service,Action,Location,Result,Cost):-
      time(1,U),U=(_,Start),
user_profile(User,Location,Service,Action,Privacy,Priority,QoS,Time,Mode),
       ( (compare(=,Priority,0);
          (compare(=,Priority,1),
           best_quality_model(Service,Location,A,B,C,D),
           maximum(A,B,C,D,F),compare(>,Time,F)
           )
        )->
    (assert(agenda(User,Service,Action,Location,Priority,Time)),
              Result=`in_agenda`);
         (Result=`proceed`)
       ),
       time(1,I),I=(_,End),Cost is End-Start.
```

literature and whether they can be supported by equipping ubiquitous applications and environments with meta-level control. This is a theoretical study and only questions regarding meta-level use will be discussed. The reader is referred to Publication VI for details.

Publication VI suggests adding meta-level functionality to ubiquitous learning systems and environments. Such architecture is similar to that presented in Figure 13, only the object-level tasks are different. Here, the object level tasks form the functionality of the ubiquitous learning applications: adaptation functionality, component composition and allocation, and task execution. Adaptation functionality implements context-based changes, e.g. changes of learning activities or content presentation. Component composition and allocation is responsible for making ensembles from the available units of learning applications and devices of the learning environment. Task execution is responsible for the actual execution of these ensembles. In this architecture, the meta-level monitors and controls presented object-level tasks. The performance of object-level tasks is evaluated by predefined criteria like student performance.

Generally, the meta-level could facilitate self-adaptation of ubiquitous learning systems by using different algorithms for object-level tasks depending on the context. Learners may receive better personalization. For instance, the meta-level could command an object-level task to use a personal mobile phone instead of a wall display for a particular learner, if this learner demonstrates poor performance when his answers are visible to others. Instructors can benefit from meta-level functionality due to its monitoring facilities. For instance, instructors could learn what works best for students and in which situations. Meta-level functionality can

include administration and maintenance tasks of ubiquitous learning applications. For instance, developers can define adaptation rules with GUI tools on how to control the application's set up based on its use. Researchers could get tools to recognise interesting patterns from performed learning activities. Overall, implementing the ideas of Publication IV to ubiquitous learning systems envisions useful functionality. However, a more detailed evaluation of how the meta-level framework can be embedded into ubiquitous learning systems is required.

A real prototype was implemented with Publication VII, which presents the Driving coach system fusing diverse real information, such as weather, driving behaviour and spatial information, information about traffic on the roads, user experience and feedback. Driving coach provides personalized recommendations regarding fuel-efficient driving. In this manuscript, only questions regarding the meta-level framework in Driving coach are covered, for a more detailed description the reader is referred to Publication VII.

Publication VII suggests reference architecture for context-aware driving systems (Figure 14). This reference architecture directly follows from the meta-level control framework presented in Section 3.3.1. As can be seen from Figure 14, only object-level tasks are different because real needs for tasks depend on the application domain and the actual scenario.

The Driving coach system architecture is presented with Figure 15. The system implements the reference architecture as follows: Data suppliers (Perceptors) and Client applications (Actuators) form the Ground level. Storage gathers Policies and profiles, Safety constraints, and History (Figure 14). Trip evaluation, Comment generation, and Model learning belong to both the Object and Meta levels. Agenda is not implemented because the only event triggering system decision making is a new trip made by a driver, which is analysed immediately. The Waiting queue is not implemented as all the required context is directly available.

As can be seen from Figure 15, the object level tasks include aggressive driving evaluation, route evaluation, fuel-efficient driving behaviour evaluation, comment generation based on fuel-efficient driving evaluation, and fuel consumption prediction. This functionality is achieved by processing the available context information. For instance, aggressive driving and fuel-efficient driving behaviour evaluation use specific procedures (details are given in Publication VII) on data retrieved from a device connected to an OBDII diagnostic connector. To

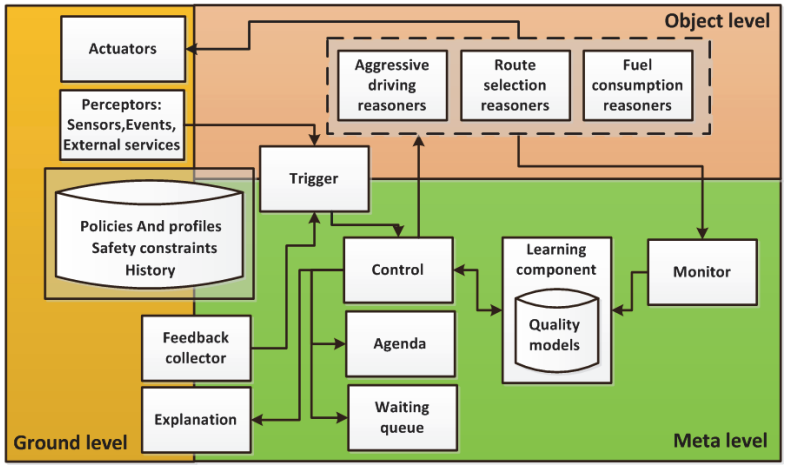

Fig. 14. Reference architecture for context-aware driving assistance systems (VII, © 2015 The Authors, CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)).

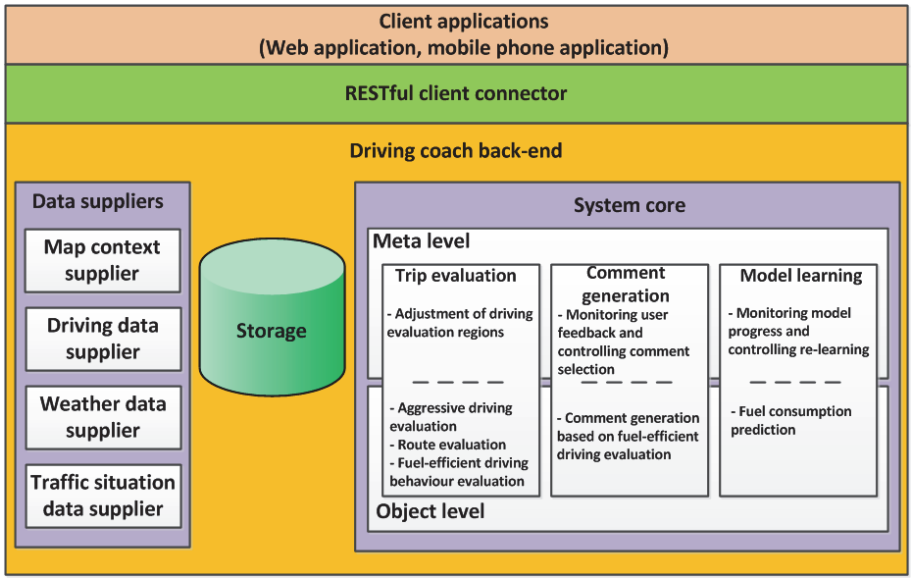

Fig. 15. Driving coach architecture (VII, © 2015 The Authors, CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)).

evaluate the route, a specific procedure is used to retrieve the spatial information from a digital map. Comments are generated with a rule-based system, encoding which advices to give for certain driving behaviour. Fuel consumption prediction uses models learned with machine learning techniques. These models use the driving behaviour and spatial information to recognize the factors affecting fuel use. Table 12 presents some examples of object-level rules.

Meta-level tasks monitor and control the execution of object level tasks as follows: first, the meta level controls fuel-efficient driving behaviour evaluation. The meta-level uses weather information and a fuzzy logic rule-based system to adjust trip evaluation. This is done to reflect the evaluation procedure on the real world. For instance, driving at certain speeds on a very slippery road is dangerous, however for normal weather conditions this is not the case. The meta-level considers such issues. Also, the meta-level monitors user feedback and controls the selection of fuel-efficient comments to be given to the driver. This enables proposing the comments which are seen to influence the driver's fuel-efficient behaviour. Table 13 demonstrates an example of the rules used to control the comment selection. The meta-level also monitors how well the fuel predicting models function and commands re-learning if required. This functionality guarantees self-adaptation, as driver behavior may change with the usage of the

Table 12. SWI-Prolog rules (simplified) generating the feedback.

Description and rule example
Defines feedback to give to the driver for each calculated factor (refer to Publication VII for details).

```
checkDriving(User):-findall(N,factor(User,N,_,_,_),R),
                    forall(member(X,R),check_factor(User,X)).

check_factor(User,N):- R is random(2),
( bad_weather(User,_)->(
   (factor(User,N,X,Y,4),X>=Y)->(advice(Z,N,4,bw,R,G),
                                 assertz(feedback(User,Z)));
   (factor(User,N,X,Y,4),X<Y)->(advice(Z,N,3,bw,R,G),
                                assertz(feedback(User,Z)));
   (factor(User,N,X,Y,3),X>=Y)->(advice(Z,N,3,bw,R,G),
                                 assertz(feedback(User,Z)));
   (factor(User,N,X,Y,3),X<Y)->(advice(Z,N,2,bw,R,G),
                                assertz(feedback(User,Z)));
   (factor(User,N,X,Y,2),X>=Y)->(advice(Z,N,2,bw,R,G),
                                 assertz(feedback(User,Z)));
   (factor(User,N,X,Y,2),X<Y)->(advice(Z,N,1,bw,R,G),
                                assertz(feedback(User,Z)));
    factor(User,N,_,_,1)->(advice(Z,N,1,bw,R,G),
                           assertz(feedback(User,Z))));
   \+(bad_weather(User,_))->(
    factor(User,N,_,_,L)->advice(Z,N,L,gw,R,G),
                          assertz(feedback(User,Z))
).
```

system.

The following results were obtained from the studies: the simulation, theoretical study and Driving coach prototype revealed the feasibility of separating the adaptive functionality to object and meta levels. The advantage of such a design is a clear separation between adaptation and control tasks [149], [32]. This provides maintenance advantages for the system, as each layer encapsulates the tasks it is responsible for. Another advantage provided is the possibility of tailoring the system to different environments and users without changes in the core Object-level tasks. For this, meta-level rules can be set for controlling the object level tasks. Also, the Meta-level layer can be shared between different systems in a ubiquitous environment, improving reusability. Finally, meta-level provides good support for self-adapting functionality of ubiquitous systems and smart spaces. Both simulation study and the Driving coach prototype demonstrated self-adaptive functionality to better serve the user. On the other hand, an additional layer introduces complexity, communication overhead, and time delays. Hence, when near real-time responses are required, careful design decisions about implementing meta-level control should be made.

The centralized solution suggested by the Meta-level control framework forces developers to use the shared context model. Such a solution on the one

Table 13. SWI-Prolog rules telling if the feedback should be changed (VII, © 2015 The Authors, CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)).

Description and rule example

Defines if the feedback should be replaced with the one having better immediate and week response (see Publication VII for details).

```
checkFeedbacksGiven(User):-findall(N,feedback(User,N),R),
                           forall(member(X,R),check_feedback(User,X)).

check_feedback(User,Id):-
N is random(3),
((N<2)->(feedback(User,Id),
  (
    score(User,Id,ImmediateResponse,WeekResponse)->(
                  advice(Id,Factor,Level,Weather,_,_),
    ((ImmediateResponse<0,WeekResponse<0)->
                                        (retract(feedback(User,Id)),
              findall(O,advice(O,Factor,Level,Weather,_,_),R),
                               find_replacement(User,R,_,NewId),
                                assertz(feedback(User,NewId),!);
true));
\+(score(User,Id,ImmediateResponse,WeekResponse))->true));
(N>1)->true
).
```

hand simplifies the development of both object-level and meta-level tasks. On the other hand, it possesses some disadvantages of a centralized approach, like scalability. One solution to overcome this problem is to equip each application with

computational reflection functionality [118], however other issues arise, like application coordination [157], [158]. Hence, more research is needed on how to achieve the fully plug-in solution for meta-level functionality for the deployed smart space.

Rule-based reasoning was used on both the object and meta levels in the prototypes. The Driving coach prototype also used machine learning to develop fuel prediction models. Use of rules for meta-level control appeared to be convenient, especially because the context model was shared between both the object and meta levels. The prototypes implement static adaptation, this means that the rules for the meta-level were constructed in advance and were not modified during system execution. The advantages outlined in Sections 2.3 and 3.1 apply here as well: rules can be changed without system recompilation. Dynamic adaptation of rules can be achieved by utilizing machine learning techniques [68].

The proposed conceptual framework is general enough to be used in different application domains. The first simulation study used a smart space as the application domain. Publication VI concentrated on ubiquitous learning. The Driving coach system focused on the driver assistance application domain. All these studies use the same conceptual framework; however, they tailor object-level tasks to concrete application domains.

4 Discussion

This thesis addresses the topic of rule-based reasoning in the ubiquitous computing domain. First, the author presented the motivation and the research problems for this work and their relevance in the ubiquitous computing domain. Second, a literature review was given to introduce the concepts used in this thesis and to position the research. Third, the research regarding the research problems was presented and analysed.

4.1 Revisiting the research questions

This thesis is guided by three research questions. This chapter revises the conducted research as well as the main contributions with respect to these research questions.

Research question 1. *What benefits and constraints does rule-based reasoning bring to the development of ubiquitous applications?*

The ubiquitous computing domain raises challenges for the development, deployment and usage of ubiquitous applications. In order to understand whether rule-based reasoning mechanisms have benefits over conventional approaches in ubiquitous computing, two similar ubiquitous applications were compared qualitatively and quantitatively. In addition, three engine implementations for these applications were compared with respect to performance and resource consumption (Publication I). Research exists on the advantages and challenges of rule-based reasoning in general [54],[55], as well as on the evaluation of the performance of reasoning engines [14],[15]. The research reported in Publication I enriches this existing scientific knowledge with its analysis of implementing application functionality with rule-based reasoning compared to a hard-coded approach, as well as comparison of three reasoning engines with a real ubiquitous computing prototype. The developed prototype uses rule-based reasoning to implement policies, and to trigger adaptive actions, similarly to [10]. For the system reported in Publication I, the reasoning engine brought scalability and maintenance advantages to the system. For instance, application functionality can be modified without system recompilation as rules are isolated from the compiled application. These findings are quite general and mostly confirm existing research [55]. From a system performance point of view, the communication overhead introduced by integrated reasoning components did not slow down the system significantly, even though additional communication is required. Hence, in our case, the decision to use rule-based reasoning was beneficial. However, as always when new methods

are taken into use, one should be aware of the extra time required to learn coding with rules.

In order to evaluate the scope of rule-based reasoning, two cases were explored. The first case studied usage of rules for a ubiquitous campus scenario [34]. In this system, rules are used to trigger adaptive actions, as in [10], [63]. The study demonstrated the importance of careful context model design. Context models for large-scale scenarios should allow easy integration of new domains and services. Also, the centralized approach for context modelling and reasoning requires attention to quality requirements, like privacy. Finally, implementation of hybrid solutions is recommended in large-scale scenarios because of the complexity and high heterogeneity of resources, for instance, to use both ontological and statistical reasoning [159]. The second case studied rule-based reasoning with mobile phones (Publication III). Research exists on the development of lightweight reasoning systems [16],[17]. Publication III contributes to scientific knowledge by exploring challenges associated with implementing reasoning on resource-constrained devices in a real ubiquitous application. This study illustrated the need for resource preserving strategies to be used when performing rule-based reasoning directly with mobile phones. Also, due to limited battery life, actions supporting knowledge base consistency should be performed. The findings reveal that rule-based reasoning can be used in a wide scope of ubiquitous applications if the capabilities and limitations of the actual smart space, as well as their match to the requirements of the concrete ubiquitous system, are carefully considered by the developer.

In order to answer to question on whether reasoning facilitates interaction in smart spaces, three different ubiquitous applications were developed and analysed. These applications have different reasoning component architectures, from fully centralized to distributed ones (Publication V). Similarly to other prototypes developed for Research question 1, rule-based reasoning in this study was responsible for policy definitions and triggering adaptive actions. Support of interaction in smart spaces is an actively researched topic [160], [161], [162], [163]. The research presented in Publication V contributes to existing scientific knowledge by identifying the actors interacting in smart spaces, and the different types of interaction. Not only is a human considered as an actor in Publication V, but also an object of a smart space is an actor if this object acts on its own and makes decisions. Finally, the analysis is performed to identify the advantages and disadvantages of each reasoning component architecture. Moreover, the effect of the reasoning component architecture on supporting the different types of interaction is explored. This study confirmed that different factors need to be

considered when designing application architecture, like the requirements of the application, the available resources and the infrastructure of the smart space. However, the conducted research demonstrated that a distributed reasoning approach supports privacy, may reduce communication overhead and supports all types of smart space interaction listed in Publication V.

This part provided mostly general findings which needed to be justified in order to gain expertise in rule-based reasoning, and its strengths and weaknesses in ubiquitous computing. The experience gained and the results of this work were used in the later stages of the research.

Research question 2. *How can rule-based reasoning be used to create ubiquitous applications for smart spaces?*

This research question explores the capabilities of rule-based reasoning to support the creation of ubiquitous applications. First, a real prototype was implemented to explore how rules can be used to control manual application composition with a physical user interface (Publication III). This system has a distributed reasoning mechanism, which uses rules to define which resources of a smart space can be used to create the application, and in which user situations. The composition of ubiquitous services is a large research field [85],[86],[80]. The conducted research contributes to interactive application composition studies [79],[81] where users are involved in the application composition process. The research demonstrates the flexibility of rules to support the creation of ubiquitous applications with physical user interfaces. Moreover, rules are understood by the users; users were able to compose the application by following the rules and manipulating the objects of the environment. Hence, the suggested approach supported a feeling of being in control.

Creating ubiquitous services with middleware support utilizing rule-based reasoning was also explored (Publication II). In order to evaluate the middleware, two similar ubiquitous services were developed and analysed, one with middleware support, the other from the scratch. The proposed middleware solution speeds up the development of ubiquitous services, by reducing the developer's efforts and taking responsibility for some maintenance issues. Rule-based reasoning is used to construct the logic of the ubiquitous service. Additional effort is required from the developers to specify context dependencies manually with a predefined vocabulary; however, such an approach provides freedom in manipulating the available contexts. It is an open issue, relevant also for related work [108], [110], how much delay is introduced by the overall intercommunication between the components of the framework. Although, a similar approach to define adaptive

behaviour with rules is found in some related work [109], [38], the solution proposed in this thesis differs conceptually and does not restrict developers with a certain rule language.

Research question 3. *How can rule-based reasoning be used to support the self-adaptation of smart spaces?*

This research question is related to the capabilities of rule-based reasoning to support self-adaptation functionality in smart spaces. In this thesis, this research problem was explored from the metareasoning perspective [32],[150],[151]. The meta concept has been applied to agent research [152] and machine learning [154]. This thesis applies it to smart spaces and their applications. In terms of ubiquitous computing and smart spaces, metareasoning is the analysis of how well the decisions and actions of a smart space support users in their tasks.

A theoretical framework for meta-level control for smart spaces and their applications was developed, and it was verified with a simulation study (Publication IV). Similarly to the MAPE-K loop reference model [118], the proposed framework locates management tasks at a separate layer. A simulation confirmed the feasibility of using rules to implement metareasoning, as well as revealed challenges for embedding such a framework into existing smart space infrastructures due to strong dependencies on the context model and interfaces. Publication VI explored the use of the framework in ubiquitous learning. Further analysis was performed based on a real prototype implementation (Publication VII). In this work, the framework was used as a carcass to implement a driving assistant system. This system fuses diverse real information to provide recommendations to a driver on how to improve the driving. These studies demonstrated feasibility in dividing the adaptive functionality into object and meta levels. Such division facilitated the maintenance and self-adapting functionality, however, with the price of additional complexity. For instance, meta-level support enabled Driving coach to adapt decision-making based on driver progress and feedback. In other words, it facilitated implicit personalisation [129]. Rule-based reasoning, used for meta-level control in both systems, implemented self-adaptive functionality in the developed prototypes. The proposed framework is general, as it could be used in different domains as Publications IV, VI, VII reveal.

To conclude the revision of research questions, Table 14 lists the advantages and disadvantages observed in the conducted research related to utilizing rule-based reasoning for each perspective addressed by the research questions (Figure 2).

Table 14. Observed advantages and disadvantages of rule-based reasoning.

Advantages	Disadvantages
Reasoning to implement functionality	
Clear separation of tasks and support for easy modifications, mainly due to the isolation of knowledge from functional components of the system and the modularity property, see Table 2 (Publication I, [34], V).	Possible delays due to the distributed nature of the system (Publication I) or large-scale scenarios [34].
Integration with other context representation formalisms ([34], Publication V).	
Wide application scope ([34], Publication III).	Maintenance challenges for large-scale [34] and resource constrained scenarios (Publication III), mainly regarding the size and consistency of the knowledge base.
Support for reasoning distribution, leading to useful qualities like privacy support, reducing communication overhead, and enriching smart space interaction (Publications III,V).	
Reasoning to create ubiquitous applications	
Supports the creation of context-aware composite applications and services (Publications II, III).	Maintenance challenges due to the complexity of interactions of users in the smart space (Publication III), mainly regarding the consistency of the knowledge base.
Supporting easy modifications, mainly due to isolation of knowledge from the functional components of the system and modularity property, see Table 2 (Publications II, III).	Requires developers to formalize application (service) functionality thoroughly at the design time (Publications II, III).
Enriches the physical interaction experience of users with smart space and due to familiarity with rule concepts facilitates the feeling of being in control (Publication III).	Delays could be observed due to communication links introduced by highly distributed middleware (Publication II).
Flexible instrument for developers to create service logic by utilizing the proposed middleware components, reducing the efforts of creating ubiquitous services (Publication II).	
Supports developers in selecting rule language (Publication II), by supporting RIF.	
Reasoning and self-adaptation in smart spaces	
Support for implementing self-adaptation in smart spaces and both object and meta level functionality (Publications IV, VII).	Dependence on context model/interfaces between meta-level and object-level (Publications IV, VII).
Tailoring of the system to certain needs and domains facilitated by the possibility to change the rules for each level independently (holding the context model dependencies are respected) (Publications IV, VI, VII).	Meta-level can introduce complexity, communication overhead, and time delays (Publications IV, VI, VII) (However, this disadvantage is mostly related to the addition of extra layer, not to the technology used to implement it).
Supporting easy modifications mainly due to isolation of knowledge from functional components of the system and modularity property, see Table 2 (Publication IV, VII).	

4.2 Main contributions

This section summarizes the main contributions of this thesis.

Demonstrative cases for using rule-based reasoning from different perspectives in smart spaces. At the moment of writing, the author is unaware of any other similar research aiming to summarize the possible use of rule-based reasoning in smart spaces. This thesis presents a holistic view on reasoning from three perspectives: use of rule-based reasoning as a component of ubiquitous applications, use of rule-based reasoning to create ubiquitous applications, and use of rule-based reasoning to achieve self-adaptation.

Publications I, III, and V contribute to the "reasoning to implement functionality of ubiquitous applications" perspective. These articles present the advantages and drawbacks of a rule-based reasoning component from qualitative and quantitative points of view (Publication I). In addition, these publications address the application scale, as well as challenges and benefits of distributing reasoning (Publications III, V). Related research covers some of these aspects [55], [54]. The articles presented in this thesis contribute through analysis that is based on the experience gained during design, implementation, and testing of real ubiquitous computing prototypes (Publications I, III, V). Generally, rules facilitate isolation of application logic from the compiled application. However, not every domain can be easily presented with rule formalisms. Moreover, rule-based reasoning can be utilized in both large scale scenarios [34] and on resource constrained devices with careful design (Publications III). Finally, distributed reasoning appears to be more flexible in terms of support of different types of interaction (Publication V).

Publications II and III contribute to the "reasoning to create ubiquitous applications" perspective. These papers demonstrate concepts, as well as real prototypes, to compose ubiquitous services and applications from elementary services by using rules. Publication II focuses on developers of such services, and proposes a middleware to develop ubiquitous services. Publication III is focused on users. These studies reveal the flexibility of rule-based reasoning to support the creation of ubiquitous applications with physical user interfaces (Publication III), as well as with middleware which dynamically selects services and data based on the rules written by the application developers (Publication II).

Finally, Publications IV, VI, and VII contribute to the "reasoning and self-adaptation in smart spaces" perspective. Publication IV provides the motivation, introduces the concepts and suggests the solution which is verified with simulation.

In the conceptual framework, adaptive functionality is divided into two levels. The simulation study uses rules to implement this functionality. Publication VI explores how this conceptual framework can be used in ubiquitous learning. Publication VII develops the ideas further with a real prototype implementation. This system implements the reference architecture and verifies the flexibility of rules to achieve self-adaptive functionality. This study reveals the flexibility, as well as the challenges with use of rule-based reasoning to implement meta-level functionality.

This thesis provides a solid reference, demonstrating the capabilities of rule-based reasoning for ubiquitous applications in smart spaces. Advantages, disadvantages, challenges, as well as design solutions to overcome them are suggested and analysed.

Demonstrative cases for using rule-based reasoning at different scales and with different architectures. At the moment of writing, the author is unaware of any other similar research aiming to analyse the scale of applying rules in the ubiquitous computing domain, as well as how and whether the architecture of a rule-based system affects the services provided by ubiquitous applications.

These aspects are explored with Publications III and V. The use of rule-based reasoning for a large scale scenario was also explored [34]. A layered context model is considered convenient for large-scale scenarios. Unfortunately, due to financial and time limitations, a simulation was built instead of a real prototype. Implementing the large scale scenario revealed the importance of the context modelling step. Publication III explores the opposite scale; it implements rule-based reasoning on a resource-constrained mobile phone. Some solutions to address context model consistency, network connectivity issues, and limited storage are presented based on this work.

Distributed reasoning is a large research topic in itself and there is a growing research interest in it, especially in the Internet of Things (IoT) research community [164], [165]. Publication V contributes to this vision. It compares several prototypes developed with different reasoning architectures. Generally, system architecture depends on many issues, like available resources and the infrastructure of the targeted smart space. The research revealed that distributed reasoning provided benefits to the developed systems, like privacy preservation, however at a cost of more difficult and time-consuming implementation.

Middleware solutions and frameworks. This thesis presents a novel middleware solution to create context-aware Web services by using rules (Publication II). Even though a large body of research exists on middleware supporting the creation of ubiquitous applications [21],[98],[99], the proposed

solution differs in some architectural and functional aspects, e.g. it supports RIF [56]. The proposed middleware is developed for ubiquitous service developers, to simplify the process of the creation of context-aware services by providing maintenance and implementation functionality. The middleware provides a rule-based mechanism to incorporate available context sources. Performance analysis revealed that it is an open issue whether the middleware can be recommended for time-sensitive ubiquitous services. However, the middleware simplifies the development work considerably and can be recommended when it can be shared among several ubiquitous services.

Another unique solution proposed by this thesis is the framework for meta-level control support for smart spaces. Publication IV introduces the framework, as well as verifies it with a simulation. Publication VI explores the application of the framework in ubiquitous learning. A real prototype reported in Publication VII took the framework as a reference. Research revealed that meta-level control supports self-adaptation of smart spaces and their applications. For instance, the driving assistant system reported in Publication VII is able to adapt itself to serve the user better in a changing context. The presented framework is general and can be used in different application domains (Publications IV, VI, VII).

Real implementations, simulations, and verifications for the introduced concepts. The research presented in this thesis is mainly based on evaluation, comparison, and analyses of the real prototypes and simulations (Publications I–V, VII, [34]). This instills more confidence in the obtained research results. The publications discuss the implementation details of the prototypes (Publications I, II, III, V, VII) and simulations (Publication IV, [34]), as well as the challenges and thoughts on how to overcome them. This generates new knowledge on how to create better, more useful and efficient ubiquitous applications and smart spaces. Some prototypes are advanced ubiquitous systems, having unique features, for instance Driving coach reported in Publication VII. Even though there is a large body of research on driver assistance systems [166], [167], Driving coach is unique in combining fusion of on-board and real-time information from third party services, identification of personal driving factors affecting the fuel use in certain situations, and adaptation of system's decision-making based on the driver's progress and feedback.

4.3 Open issues

There are some issues requiring attention, though they were not within the focus of this thesis. First, the presented research does not address in detail context inconsistency and ambiguity [9]. Ubiquitous systems operate "in the wild" in partially observed environments and among inhabitants with unpredictable behaviour. In such circumstances, it is often difficult to get the right and precise context information. The basic approach to tackling imprecise context information is to use probabilistic methods, like probabilistic logic and Bayesian networks [53]. Most of the research of this thesis presents cases where all the details required by the application can be observed, and the recognized context is known and valid. This approach was selected because the developed prototypes aimed to verify the introduced theoretical concepts. Hence, the prototype scenarios do not have a very large scope, and do not use imprecise context information. The last prototype required more work to prepare context information, as this prototype works with real data (Publication VII). However, the frameworks presented in the thesis allow the inclusion of functionality to use imprecise context, e.g with an additional probabilistic module or as a part of context suppliers (Publication II, IV).

Another aspect deserving attention, not, however, addressed in this thesis in detail, is evaluation of smart space applications and frameworks. Evaluation of ubiquitous computing applications is challenging. Due to the costs for real deployment, time constraints, and ethical issues, most evaluations were conducted on a smaller scale, with controlled user studies. Laboratory user studies are good to check whether the developed technology works as expected, to observe user behaviour, and to find the usability problems; however, they are limited in grasping the usage patterns in real world situations [168], [26]. Field studies could give a deeper understanding of technology usage. Our prototypes were analysed with laboratory user studies and simulations. This gave us full control over the situation, as the purpose of the developed prototypes was to verify the concepts, and not to understand human-computer interaction patterns.

Evaluation of ubiquitous computing frameworks and middleware is even more challenging, because this also includes evaluation of how well the middleware supports developers and how difficult it is to maintain. For instance, Ranganathan et al. [169] summarize their experience with the Gaia system into an evaluation benchmark, consisting of three layers of parameters: system metrics, configurability and programmability metrics, and human usability metrics. One of more recent taxonomy-based approaches to evaluate ubiquitous systems is

presented by Yang et al. [170]. Our Perception Framework middleware (Publication II) was evaluated with standard performance criteria. The meta-level control framework represents a reference architecture, which was verified first with simulation, and then with a real prototype. Both the simulation (Publication IV) and the prototype (Publication VII) do not address performance issues.

The final issue we would like to outline is privacy in ubiquitous computing. Generally defined as "the right to be let alone", privacy supports individuals, groups, and institutions to define by themselves which information, when, with whom, and how to communicate [30], [31]. Ubiquitous computing assumes massive sensing, storage, and transfer of data, and hence raises privacy issues. An example of privacy violation is to perform reasoning over information one does not want to share. Privacy in ubiquitous computing is a wide research topic itself. For instance, Schaub et al. [171] explore the effect of context in privacy expectations of users and dynamic privacy regulation with respect to social interaction. Our research is not focused on privacy support. However, we have touched this topic in two studies (Publications III, V). The QuizBlasters prototype makes most of its reasoning on a user's mobile phone, hence it does not send personal information over the network. The same behaviour is observed for the SI prototype. This makes the QuizBlasters and SI prototypes privacy preserving.

4.4 Future work

A number of potential research directions follow from the research presented in this thesis. First, the ideas proposed in this research can be applied to other domains. For instance, it would be very interesting to implement a real prototype and analyse the effect of equipping the learning applications with meta-level functionality (Publication VI).

Some future ideas are related to the IoT [172], [174], [173]. Future environments are predicted to be full of tiny, computationally enriched devices supporting interaction between humans and these devices (things). It is necessary to develop lightweight mechanisms to encode semantic knowledge, lightweight communication protocols, and efficient processing and reasoning algorithms. These would enable these tiny little devices to talk to each other and build up a knowledge base gradually, as humans do, and to achieve truly resource-to-resource and facilitated interaction, as mentioned in Publication V. Moreover, research is needed to understand how the smart space should be organized physically to best enable interactions, e.g. how to overcome the invisibility dilemma [18]. If we

would be able to make the interaction between things and people social, personalized, proactive, and predictable, we would take a step further from the IoT to the Internet of People (IoP) [143].

In addition, these small devices will produce massive amounts of sensor data, which should be mined and reasoned by utility services. Handling such data requires specific solutions for online reasoning [175], [164]. It would be interesting to explore how the middleware for ubiquitous service composition (Publication II) can be adapted to these very large scale needs.

Most of the research presented in this thesis demonstrates centralized reasoning solutions. However, the IoT vision, the volatile nature of smart spaces, as well as multiuser interactions, dictate the need to move to an asynchronous and decentralized reasoning approach. This thesis also presents the advantages of distributed reasoning over a centralized approach for smart space interaction. It would be interesting to explore how the proposed meta-level control framework (Publications IV, VI, VII) can be integrated into distributed IoT scenarios with massive data production.

5 Conclusions

We are witnessing how technology is becoming interwoven in our everyday lives and changing our expectations about what computers are and what they might do for us [176]. As a result, people can concentrate on their daily tasks by relying on technology as an assistive tool. The ubiquitous computing paradigm emphasizes the importance of context in application development. Ubiquitous applications are able to recognize the context and react accordingly. Different approaches have been suggested to implement the decision-making processes of ubiquitous applications. Rule-based reasoning approach follows Ligeza's [13] statement: "*Thinking in terms of facts and rules is perhaps one of the most common ways of approaching problem definition and problem solving both in everyday life and under more formal circumstances.*"

This thesis explored the use of rule-based reasoning in ubiquitous computing. Particularly, three perspectives on the use of rule-based reasoning were studied. First, research was conducted towards understanding what rule-based reasoning brings to smart space applications in general. When compared to the hard-coded approach for implementing ubiquitous application functionality, the use of rule-based reasoning brings maintenance advantages by isolating rules from the compiled application. Also, the scope of rule-based reasoning was explored. Designing large-scale ubiquitous applications requires paying attention to the context model. An ontology-based context model with a layered approach appeared to be flexible allowing also hybrid reasoning techniques. On the other hand, implementing rule-based reasoning on resource constrained devices requires actions to support consistency, due to limited resources. In addition, the effect of reasoning component architecture on different types of interaction in smart spaces was studied. Analysis of the prototypes with different architecture for the reasoning component produced the conclusion that the distributed solution provides richer interaction capabilities for smart spaces. Also, the distributed approach provides some advantageous features, such as processing privacy-sensitive data in the users' own mobile devices.

Second, research was performed to understand how rule-based reasoning can be used to support the creation of ubiquitous applications, considering both users and developers of ubiquitous services. Rule-based reasoning is found to be useful to support both users and developers to create ubiquitous applications. First, it supports users in composing ubiquitous applications by providing the familiar rule metaphor for defining the use of resources of a smart space. Second, it appears that

rules are a convenient means to describe application logic for services developed with the middleware-based approach. With this approach developers can use rules and low-level contexts to construct high-level contexts and to define adaptive behaviour for ubiquitous services. This way, developers can concentrate on the application logic and leave the underlying complexity of actual implementation of low-level contexts and the interconnection between components for the middleware.

Third, research was performed to gain understanding of how rule-based reasoning can support self-adaptation. Here, the role of rules to implement metareasoning functionality for ubiquitous applications was explored. Due to complex execution environments, self-introspective properties are desired for ubiquitous computing applications. This thesis proposed a meta-level control framework which adds control and monitoring functionality to ubiquitous applications and smart spaces. This framework was verified with a simulation and with theoretical studies, as well as a real world prototype. The research demonstrated the usefulness of using rule-based reasoning to implement the meta-level of the proposed framework. Moreover, both the simulation and the prototype demonstrated self-adaptive functionality, proving the applicability of the meta-level control to support self-adaptation for smart spaces and their applications. Applying the framework presented to different domains indicates its generality.

This thesis contributed to the creation of a holistic view on the challenges, benefits, and capabilities of rule-based reasoning in building ubiquitous applications. The results presented in this thesis can be used as guidelines for the development of ubiquitous applications. Finally, the author hopes that this thesis will inspire future research on ubiquitous computing and related fields.

List of references

1. Dey AK (2001) Understanding and Using Context. Int J Personal Ubiquitous Computing 5(1): 4–7.
2. Nurmi P & Floréen P (2004) Reasoning in Context-Aware Systems. Position paper, Helsinki Institute for Information Technology (HIIT),Basic Research Unit (BRU).
3. Poslad S (2009) Ubiquitous computing: smart devices, environments and interactions. John Wiley & Sons Ltd.
4. Weiser M (1999) The computer for the 21st century. SIGMOBILE Mob Comput Commun Rev 3(3): 3–11.
5. Cook DJ & Das SK (ed) Smart environments: Technology, protocols, and applications, John Wiley & Sons, Inc., Hoboken, NJ.
6. Ye J, Dobson S & McKeever S (2012) Situation identification techniques in pervasive computing: A review. Pervasive and Mobile Computing 8(1): 36-66.
7. Wei EJY & Chan ATS (2007) Towards Context-Awareness in Ubiquitous Computing. In: Kuo TW et al. (ed) Embedded and Ubiquitous Computing. LNCS, Springer, 4808: 706–717.
8. Viterbo J (2009) Decentralized Reasoning in Ambient Intelligence. Doctoral Thesis, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).
9. Henricksen K & Indulska J (2004) Modelling and using imperfect context information. Proc IEEE Pervasive Computing and Communications Workshops, 33–37.
10. Su X, Gilman E, Kwiatkowski P, Latkowski T, Pröbstl A, Wojtowicz B & Riekki J (2011) Knowledge-based Systems for Ambient Social Interactions. In: Keyson DV et al. (ed) Ambient Intelligence. LNCS, Springer, 7040:61–71.
11. Perttunen M, Riekki J & Lassila O (2009) Context representation and reasoning in pervasive computing: a review. Int J of Multimedia and Ubiquitous Engineering, 4(4):1-28.
12. 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 6(2): 161–180.
13. Ligeza A (2006) Logical foundations for rule-based systems, Springer.
14. Liang S, Fodor P, Wan H & Kifer M (2009) OpenRuleBench: an analysis of the performance of rule engines. Proc. 18th international Conference on World Wide Web (WWW '09): 601–610.
15. Van Woensel W, Al Haider N, Ahmad A, Abidi SSR (2014) A cross-platform benchmark framework for mobile semantic web reasoning engines. In: Mika P et al. (ed) The Semantic Web – ISWC 2014, Springer, LNCS 8796: 389–408.
16. Androjena. URI: https://github.com/lencinhaus/androjena. Cited 2015/04/06.
17. Iglesias J, Bernardos AM, Tarrío P, Casar JR & Martín H (2012) Design and validation of a light inference system to support embedded context reasoning. Pers Ubiquit Comput 16 (7): 781-797.
18. Kranz M, Holleis P & Schmidt A (2010) Embedded interaction : interacting with the internet of things. IEEE Internet Computing 14(2):46–53.

19. Xiao H, Zou Y, Tang R, Ng J & Nigul L (2010) A framework for automatically supporting end-users in service composition. In: Chignell M et al. (ed) The Smart Internet, Springer, LNCS 6400: 115–136.
20. Gajos K, Fox H & Shrobe H (2002) End User Empowerment in Human Centred Pervasive Computing. Proc Pervasive 2002, Zurich, 1–7.
21. Perera C, Zaslavsky A, Christen P & Georgakopoulos D (2014) Context Aware Computing for The Internet of Things: A Survey, IEEE Communications Surveys & Tutorials 16(1): 414–454.
22. Bellavista P, Corradi A, Fanelli M & Foschini L (2012) A survey of context data distribution for mobile ubiquitous systems. ACM Comput Surv 44(4), Article 24.
23. Edwards WK & Grinter RE (2001) At Home with Ubiquitous Computing: Seven Challenges. In: Abowd GD et al. (ed) Ubicomp 2001: Ubiquitous Computing, Springer, LNCS 2201:256–272.
24. Vermeulen J, Vanderhulst G, Luyten K & Coninx K (2010) PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. Proc International Conference on Intelligent Environments, 271-276.
25. Bellotti V and Edwards K (2001) Intelligibility and accountability: human considerations in context-aware systems. Hum.-Comput. Interact. 16(2): 193–212.
26. Krumm J (ed) (2010) Ubiquitous computing fundamentals. CRC Press.
27. Salehie M & Tahvildari L (2009) Self-adaptive software: Landscape and research challenges. ACM Trans Auton Adapt Syst 4(2), Article 14.
28. Huebscher MC & McCann JA (2008) A survey of autonomic computing—degrees, models, and applications. ACM Comput Surv 40(3), Article 7.
29. Lee YJ, Lim J, Hyun SJ & Lee D (2010) Rule-based Approach for Context Inconsistency Management Scheme in Ubiquitous Computing. Proc IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC): 120–126.
30. Bettini C & Riboni D (2015) Privacy protection in pervasive systems: State of the art and technical challenges. Pervasive and Mobile Computing 17: 159–174.
31. Langheinrich M (2001) Privacy by design – principles of privacy-aware ubiquitous systems. Ubicomp 2001: Ubiquitous Computing, LNCS 2201:273–291.
32. Cox M & Raja A (2008) Metareasoning: a Manifesto. Proc. Metareasoning: Thinking about Thinking workshop held within 23 AAAI Conf. on Artificial Intelligence.
33. Lukka K (2003) The constructive research approach. In: Ojala L & Hilmola OP (ed) Case study research in logistics, 83–102.
34. Gilman E, Su X & Riekki J (2011) Towards Context Modelling and Reasoning in Ubiquitous Campus. Heimbürger A et al. (Eds.), Information Modelling and Knowledge Bases XXII, Volume 225 Frontiers in Artificial Intelligence and Applications: 278–287.
35. Schilit BN, Adams N & Want R (1994) Context-Aware Computing Applications. Proc Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA: 85–90.
36. Winograd T (2001) Architectures for context. Hum.-Comput. Interact. 16 (2):401–419.

37. Soylu A, De Causmaecker P & Desmet P (2009) Context and Adaptivity in Context-Aware Pervasive Computing Environments. Proc Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing (UIC-ATC '09): 94–101.
38. Zimmermann A (2007) Context Management and Personalisation: A Tool Suite for Context- and User-Aware Computing. Doctoral Thesis, Fraunhofer FIT.
39. Dey AK & Abowd GD (2000) Towards a better understanding of context and context-awareness. Presented at CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, The Nethrlands.
40. Want R, Hopper A, Falcão V & Gibbons J (1992) The active badge location system. ACM Trans. Inf. Syst. 10 (1): 91–102.
41. Asthana A, Cravatts M & Krzyzanowski P (1994) An Indoor Wireless System for Personalized Shopping Assistance. Proc Workshop on Mobile Computing Systems and Applications (WMCSA '94): 69–74.
42. Abowd GD, Atkeson CG, Hong J, Long S, Kooper R & Pinkerton M (1997) Cyberguide: a mobile context-aware tour guide. Wirel. Netw. 3(5):421–433.
43. Shih S-C, Kuo B-C & Liu Y-L (2012) Adaptively ubiquitous learning in campus math path. Educational Technology & Society 15(2): 298–308.
44. Chen G & Kotz D (2000) A Survey of Context-Aware Mobile Computing Research. Technical Report. Dartmouth College, Hanover, NH, USA.
45. Baldauf M, Dustdar S & Rosenberg F (2007) A survey on context-aware systems. Int J Ad Hoc Ubiquitous Comput. 2(4): 263–277.
46. Strang T & Linnhoff-Popien C (2004) A context modeling survey. Proc Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham/England.
47. Ye J, Dasiopoulou S, Stevenson G, Meditskos G, Kontopoulos E, Kompatsiaris I & Dobson S (2015) Semantic web technologies in pervasive computing: a survey and research roadmap. Pervasive and Mobile Computing, in press.
48. Bikakis A, Patkos T, Antoniou G, Plexousakis D (2008) A survey of semantics-based approaches for context reasoning in ambient intelligence. In: Mühlhäuser M et al. (ed) Constructing Ambient Intelligence. Communications in Computer and Information Science 11: 14–23.
49. Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (ed) (2003) The description logic handbook: theory, implementation, and applications, Cambridge University Press.
50. Predoiu L & Stuckenschmidt H (2009). Probabilistic Models for the Semantic Web: A Survey. In: Ma Z & Wang H (ed) The Semantic Web for Knowledge and Data Management, IGI Global: 74–105.
51. Aamodt A & Plaza E (1994) Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press 7(1): 39–59.
52. Lukasiewicz T, Straccia U (2008) Managing uncertainty and vagueness in description logics for the Semantic Web. Web Semantics: Science, Services and Agents on the World Wide Web 6(4): 291–308.

53. Hopgood AA (2012) Intelligent Systems for Engineers and Scientists, 3rd edition, CRC Press.
54. Bikakis A & Antoniou G (2010) Rule-based Contextual Reasoning in Ambient Intelligence. In: Dean M et al. (ed) Semantic Web Rules, LNCS, Springer, 6403:74–88.
55. Sasikumar M, Ramani S, Raman SM, Anjaneyulu KSR, Chandrasekar R (2007) A Practical Introduction to Rule Based Expert Systems, Narosa Publishing House, New Delhi.
56. Kifer M (2008) Rule Interchange Format: The Framework. In: Calvanese D & Lausen G (ed) Web Reasoning and Rule Systems, Springer, LNCS 5341: 1–11.
57. Lim BY & Dey AK (2010) Toolkit to support intelligibility in context-aware applications. Proc 12th ACM International conference on Ubiquitous computing (UbiComp'10): 13–22.
58. Cao Y, Tao L & Xu G (2009) An Event-driven Context Model in Elderly Health Monitoring. Proc Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing (UIC-ATC '09): 120–124.
59. Chen YL, Chiang HH, Yu CW, Chiang CY, Liu CM & Wang JH (2012) An intelligent knowledge-based and customizable home care system framework with ubiquitous patient monitoring and alerting techniques. Sensors 12(8):11154–11186.
60. Costa PD, Almeida JPA, Pires LF & van Sinderen M (2007) Situation specification and realization in rule-based context-aware applications. In: Indulska J & Raymond K (ed) Distributed Applications and Interoperable Systems, Springer, LNCS 4531: 32–47.
61. Toutain F, Ramparany F & Szczekocka E (2012) Semantic Context Reasoning for Formulating User Location. Proc 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA): 671–677.
62. Bonino D & Corno F (2010) Rule-based intelligence for domotic environments. Automation in Construction 19(2): 183–196.
63. Armenatzoglou N, Marketakis Y, Kriara L, Apostolopoulos E, Papavasiliou V, Kampas D, Kapravelos A, Kartsonakis E, Linardakis G, Nikitaki S, Bikakis A, and Grigoris Antoniou (2009) FleXConf: A Flexible Conference Assistant Using Context-Aware Notification Services. In: Meersman R et al. (ed) On the Move to Meaningful Internet Systems: OTM 2009 Workshops, Springer, LNCS 5872: 108–117.
64. Halpern JY & Weissman V (2003) Using first-order logic to reason about policies. Proc 16th IEEE Computer Security Foundations Workshop: 187–201.
65. Fong J, Lam HP, Robinson R & Indulska J (2012) Defeasible preferences for intelligible pervasive applications to enhance eldercare. Proc IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops): 572–577.
66. Garcia-Herranz M, Haya P & Alaman X (2010) Towards a ubiquitous end-user programming system for smart spaces. J of Universal Computer Science 16(12): 1633–1649.
67. Boyaci O, Martinez VB & Schulzrinne H (2012) Bridging Communications and the Physical World. IEEE Internet Computing 16(2): 35–43.

68. Papadopoulou E, Taylor NK, Williams MH & Gallacher S (2013) Learning user preferences in a system combining pervasive behaviour and social networking. Proc 8th International Conference on Information Technology in Asia (CITA):1–7.
69. Mitchell R & Chen IR (2012) Behavior Rule Based Intrusion Detection for Supporting Secure Medical Cyber Physical Systems. Proc 21st International Conference on Computer Communications and Networks (ICCCN): 1–7.
70. Chien BC, Hsueh YK (2011) Initiative prevention strategy for context inconsistency in smart home. Proc IEEE International Conference on Granular Computing (GrC):138–143.
71. Degeler V & Lazovik A (2011) Interpretation of inconsistencies via context consistency diagrams. Proc IEEE International Conference on Pervasive Computing and Communications (PerCom): 20–27.
72. Filho JB & Agoulmine N (2011) A Quality-Aware Approach for Resolving Context Conflicts in Context-Aware Systems. Proc IFIP International Conference on Embedded and Ubiquitous Computing (EUC): 229–236.
73. Bikakis A & Antoniou G (2008) Distributed defeasible contextual reasoning in ambient computing. In: Aarts E et al. (ed) Ambient Intelligence, Springer, LNCS 5355: 308–325.
74. Gu T, Pung HK & Zhang D (2008) Peer-to-Peer Context Reasoning in Pervasive Computing Environments. Proc IEEE International Conference on Pervasive Computing and Communications (PerCom):406–411.
75. Almeida A & Lopez-de-Ipina D (2012) An inference sharing architecture for a more efficient context reasoning. Proc IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops): 848–852.
76. Ye J, Li J, Shi H, Gu X & Zhu Z (2008) DFre: A Distributed Fuzzy Reasoning Engine for Personalization Recommendation. Proc 3rd International Conference on Pervasive Computing and Applications (ICPCA): 576–581.
77. Gross T, Paul-Stueve T & Palakarska T (2007) SensBution: A Rule-Based Peer-to-Peer Approach for Sensor-Based Infrastructures. Proc 33rd EUROMICRO Conference on Software Engineering and Advanced Applications: 333–340.
78. Bikakis A & Antoniou G (2010) Defeasible Contextual Reasoning with Arguments in Ambient Intelligence. IEEE Transactions on Knowledge and Data Engineering 22(11): 1492–1506.
79. Davidyuk O (2012) Automated and interactive composition of ubiquitous applications. Doctoral Thesis, University of Oulu.
80. Stavropoulos TG, Vrakas D, Vlahavas I (2013) A survey of service composition in ambient intelligence environments. Artif Intell Rev 40:247–270.
81. Davidyuk O, Georgantas N, Issarny V & Riekki J (2011) MEDUSA: Middleware for end-user composition of ubiquitous applications. In: Chong NY & Mastrogiovanni F (ed) Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives, IGI Global: 197–219.

82. Chinnici R, Moreau JJ, Ryman A, Weerawarana S, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. URI: http://www.w3.org/TR/wsdl20/. Cited 2015/05/06.
83. Martin D, Burstein M, Hobbs J & et al., OWL-S: Semantic Markup for Web Services. URI: http://www.w3.org/Submission/OWL-S/. Cited 2015/05/06.
84. Carey K, Lewis D, Higel S & Wade V (2004) Adaptive composite service plans for ubiquitous computing. Proc 2nd International Workshop on Managing Ubiquitous Communications and Services (MUCS 2004).
85. Dustdar S & Schreiner W (2005) A survey on web services composition. Int J Web Grid Serv 1(1): 1–30.
86. Rao J & Su X (2004) A survey of automated web service composition methods. Proc First International Conference on Semantic Web Services and Web Process Composition (SWSWPC'04): 43–54.
87. Zhou J, Gilman E, Palola J, Riekki J, Ylianttila M & Sun J (2011) Context-aware pervasive service composition and its implementation. Pers Ubiquit Comput 15(3): 291–303.
88. Mokhtar SB, Georgantas N & Issarny V (2005) Ad hoc composition of user tasks in pervasive computing environments. In: Gschwind T et al. (ed) Software Composition, Springer, LNCS 3628: 31–46.
89. Fissaa T, Guermah H, Hafiddi H, Nassar M & Kriouile A (2014) Towards an ontology based architecture for context-aware services composition. Proc International Conference on Multimedia Computing and Systems (ICMCS): 990–995.
90. Zahoor E, Perrin O & Godart Claude (2009) Rule-based semi automatic Web services composition. Proc World Conference on Services, Los Angeles, CA, 805–812.
91. Beattie M, Hallberg J, Nugent C, Synnes K, Cleland I & Lee S (2015) Smart Homes and Health Telematics, LNCS 8456: 93–102.
92. Mavrommati I, Markopoulos P, Calemis J, Kameas A (2003) Experiencing Extrovert Gadgets. Proc HCI 2: 179–182.
93. Wisner P & Kalofonos DN (2007) A Framework for End-User Programming of Smart Homes Using Mobile Devices. Proc IEEE Consumer Communications and Networking Conference (CCNC 2007): 716–721.
94. Chin JS, Callaghan V & Clarke G (2006) An End-User Programming Paradigm for Pervasive Computing Applications, Proc ACS/IEEE International Conference on Pervasive Services, Lyon: 325–328.
95. Aztiria A, Augusto JC, Basagoiti R, Izaguirre A & Cook DJ (2013) Learning frequent behaviors of the users in intelligent environments. IEEE Transactions on Systems, Man, and Cybernetics: Systems 43(6): 1265–1278.
96. Mahmoud QH (2004)(ed)Middleware for Communication, Wiley, Hoboken, NJ.
97. Schiele G, Handte M & Becker C (2010) Pervasive Computing Middleware. In: Nakashima H et al. (ed) Handbook of Ambient Intelligence and Smart Environments: 201–227.
98. Raychoudhury V, Cao J, Kumar M & Zhang D (2013) Middleware for pervasive computing: A survey. Pervasive and Mobile Computing 9(2): 177–200.

99. Knappmeyer M, Kiani SL, Reetz ES, Baker N & Tönjes R (2013) Survey of Context Provisioning Middleware. IEEE Communications Surveys & Tutorials 15(3): 1492 – 1519.
100. Saeed A & Waheed T (2010) An extensive survey of context-aware middleware architectures. Proc IEEE International Conference on Electro/Information Technology (EIT): 1–6.
101. Makris P, Skoutas DN, Skianis C (2013) A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments' Integration. IEEE Communications Surveys & Tutorials 15(1): 362–386.
102. Dey AK (2000) Providing architectural support for building context-aware applications. Doctoral Thesis, Georgia Institute of Technology, Atlanta, GA.
103. Román M, Hess K, Cerqueira R, Ranganathan A, Campbell RH & Nahrstedt K (2002) A middleware infrastructure for active spaces. IEEE Pervasive Computing 1(4): 74–83.
104. Schmidt A (2002) Ubiquitous computing - computing in context. Doctoral Thesis, Lancaster University.
105. Mitchell K (2002) Supporting the development of mobile context-aware systems. Doctoral Thesis, Lancaster University.
106. Chen HL (2004) An Intelligent Broker Architecture for Pervasive Context-Aware Systems. Doctoral dissertation, University of Maryland.
107. Korpipää P (2005) Blackboard-based software framework and tool for mobile device context awareness. Doctoral dissertation, VTT Technical Research Centre of Finland.
108. Bardram JE (2005) The java context awareness framework (JCAF) – a service infrastructure and programming framework for context-aware applications. In: Gellersen HW et al. (ed) Pervasive Computing, Springer, LNCS 3468: 98–115.
109. Gu T, Pung HK & Zhang DQ (2005) A service-oriented middleware for building context-aware services. J Netw Comput Appl 28(1):1–18.
110. Athanasopoulos D, Zarras AV, Issarny V, Pitoura E & Vassiliadis P (2008) CoWSAMI: Interface-aware context gathering in ambient intelligence environments. Pervasive Mob Comput 4(3):360–389.
111. Dogdu E & Soyer O (2013) MoReCon: a mobile restful context-aware middleware. Proc of the 51st ACM Southeast Conference (ACMSE '13), Article 37.
112. Paspallis N & Papadopoulos GA (2014) A pluggable middleware architecture for developing context-aware mobile applications. Pers Ubiquit Comput,18(5):1099–1116.
113. Maia MEF, Fonteles A, Neto B, Gadelha R, Viana W & Andrade RMC (2013) LOCCAM - loosely coupled context acquisition middleware. Proc of the 28th Annual ACM Symposium on Applied Computing (SAC '13):534–541.
114. Ferreira D (2013) AWARE: A mobile context instrumentation middleware to collaboratively understand human behaviour, Doctoral Thesis, University of Oulu.
115. Forsström S & Kanter T (2014) Enabling ubiquitous sensor-assisted applications on the internet-of-things. Personal Ubiquitous Comput 18(4): 977–986.
116. Teixeira T, Hachem S, Issarny V & Georgantas N (2011) Service oriented middleware for the internet of things: a perspective. In: Abramowicz W et al. (ed) Towards a Service-Based Internet, Springer,LNCS 6994: 220–229.

117. Krupitzer C, Roth FM, VanSyckel S, Schiele G & Becker C (2015) A survey on engineering approaches for self-adaptive systems. Pervasive and Mobile Computing 17: 184–206.
118. Kephart JO & Chess DM (2003) The Vision of Autonomic Computing. Computer 36(1): 41–50.
119. Ahmed S, Ahamed SI, Sharmin M & Hasan CS (2009) Self-healing for Autonomic Pervasive Computing. In: Vasilakos AV et al. (ed) Autonomic Communication, Springer:285–307.
120. Trumler W, Petzold J, Bagci F & Ungerer T (2006) AMUN: an autonomic middleware for the Smart Doorplate Project. Personal Ubiquit Comput 10(1): 7–11.
121. Lupu E, Dulay N, Sloman M, Sventek J, Heeps S, Strowes S, Twidle K, Keoh SL & Schaeffer-Filho A (2008) AMUSE: autonomic management of ubiquitous e-Health systems. Concurr Comput : Pract Exper 20(3):277–295.
122. Gouin-Vallerand C, Abdulrazak B, Giroux S (2009) A self-configuration middleware for smart spaces. Int J of Smart Home 3(1): 7–16.
123. Zhang W, Hansen KM & Ingstrup M (2014) A hybrid approach to self-management in a pervasive service middleware. Knowledge-Based Systems (67):143–161.
124. Mizouni R, Matar MA, Mahmoud ZA, Alzahmi S & Salah A (2014) A framework for context-aware self-adaptive mobile applications SPL. Expert Systems with Applications 41(16): 7549–7564.
125. Floch J, Fra C, Fricke R, Geihs K, Wagner M, Lorenzo J, Soladana E, Mehlhase S, Paspallis N, Rahnama H, Ruiz PA & Scholz U (2013) Playing MUSIC – building context-aware and self-adaptive mobile applications. Softw Pract Exper 43:359–388.
126. Gunasekera K, Zaslavsky A, Krishnaswamy S & Loke SW (2013) Building ubiquitous computing applications using the VERSAG adaptive agent framework. J of Systems and Software 86(2): 501–519.
127. Lasierra N, Alesanco A, O'Sullivan D & García J (2013) An autonomic ontology-based approach to manage information in home-based scenarios: From theory to practice. Data & Knowledge Engineering 87: 185–205.
128. Sousa JP, Poladian V, Garlan D, Schmerl B & Shaw M (2006) Task-based adaptation for ubiquitous computing. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 36(3): 328–340.
129. Wagner A, Barbosa JLV & Barbosa DNF (2014) A model for profile management applied to ubiquitous learning environments. Expert Systems with Applications 41(4): 2023-2034.
130. Evers C, Kniewel R, Geihs K & Schmidt L (2014) The user in the loop: Enabling user participation for self-adaptive applications. Future Generation Computer Systems 34: 110–123.

131. Gamrat C, Philippe JM, Jesshope C, Shafarenko A, Bisdounis L, Bondi U, Ferrante A, Cabestany J, Huebner M, Pärsinnen J, Kadlec J, Danek M, Tain B, Eisenbach S, Auguin M, Diguet JP, Lenormand E & Roux JL (2011) AETHER: Self-Adaptive Networked Entities: Autonomous Computing Elements for Future Pervasive Applications and Technologies. In: Cardoso JMP & Hubner M (ed) Reconfigurable Computing: 149–184.
132. Huang CH, Shen JS & Hsiung PA (2010) A self-adaptive hardware/software system architecture for ubiquitous computing applications. In: Yu Z et al. (ed) Ubiquitous Intelligence and Computing, Springer, LNCS 6406:382–396.
133. Fujii K & Suda T (2009) Semantics-based context-aware dynamic service composition. ACM Trans Auton Adapt Syst 4 (2), Article 12.
134. Sánchez I, Riekki J, Rousu J & Pirttikangas S (2008) Touch & Share: RFID based ubiquitous file containers. Proc of the 7th International Conference on Mobile and Ubiquitous Multimedia (MUM '08):57–63.
135. Riekki J, Sanchez I & Pyykkonen M (2010) Remote control for pervasive services. Int J Auton Adapt Commun Syst 3(1): 39–58.
136. Paganelli F, Bianchi G & Giuli D (2007) A Context model for context-aware system design towards the ambient intelligence vision: experiences in the eTourism domain. In: Stephanidis C & Pieper M (ed) Universal Access in Ambient Intelligence Environments, Springer, LNCS 4397: 173–191.
137. Lee D & Meier R (2009) A hybrid approach to context modelling in large-scale pervasive computing environments. Proc of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE '09), Article 14.
138. Su X, Riekki J & Haverinen J (2012) Entity Notation: enabling knowledge representations for resource-constrained sensors. Personal Ubiquitous Comput 16(7):819–834.
139. Peters M, Brink C, Sachweh S & Zündorf A (2014) Scaling parallel rule-based reasoning. In: Presutti V et al. (ed) The Semantic Web: Trends and Challenges, Springer, LNCS 8465:270–285.
140. Koch F, Meyer JJC, Dignum F, Rahwan I (2005) Programming deliberative agents for mobile services: the 3APLM platform. Proc of the AAMAS'05 Workshop on Programming Multi-Agent Systems (ProMAS'05): 222–235.
141. Alves P & Ferreira P (2011) Distributed Context-aware systems. Technical report RT/22/2011.
142. Nalepa GJ, Bobek S (2014) Rule-Based Solution for Context-Aware Reasoning on Mobile Devices. Computer Science and Information Systems 11(1): 171–193.
143. Miranda J, Mäkitalo N, Garcia-Alonso J, Berrocal J, Mikkonen T, Canal C & Murillo JM (2015) From the Internet of Things to the Internet of People. IEEE Internet Computing 19(2): 40–47.
144. Wernke M, Skvortsov P, Durr F & Rothermel K (2014) A classification of location privacy attacks and approaches. Pers Ubiquit Comput 18:163–175.

145. Jeng T (2009) Toward a Ubiquitous Smart Space Design Framework. J of Information Science and Engineering 25: 675–686.
146. Vermeulen J (2014) Designing for intelligibility and control in ubiquitous computing environments, Doctoral Thesis, Hasselt University.
147. Kaasinen E, Kymäläinen T, Niemelä M, Olsson T, Kanerva M & Ikonen V (2013) A User-Centric View of Intelligent Environments: User Expectations, User Experience and User Role in Building Intelligent Environments. Computers 2(1): 1–33.
148. Davidyuk O, Sanchez I & Riekki J (2011) CADEAU: Supporting Autonomic and User-Controlled Application Composition in Ubiquitous Environments. In: Malatras A (ed) Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications, Information Science Reference, IGI Global, 74–102.
149. McKinley PK, Sadjadi SM, Kasten EP & Cheng BHC (2004) Composing adaptive software. Computer 37(7): 56–64.
150. Cox M & Raja A (ed) (2011) Metareasoning: Thinking about Thinking. The MIT Press.
151. Raja A (2003) Meta-Level Control in Multi-Agent Systems. Doctoral Thesis, University of Massachusetts Amherst.
152. Cannella V, Chella A & Pirrone R (2013) A meta-cognitive architecture for planning in uncertain environments. Biologically Inspired Cognitive Architectures 5:1–9.
153. Vilalta R & Drissi Y (2002) A perspective view and survey of meta-learning. Artificial Intelligence review 18(2): 77–95.
154. Anderson ML & Oates T (2007) A review of recent research in metareasoning and metalearning. AI Magazine 28(1): 7–16.
155. Serbedzija N & Fairclough S (2012) Reflective pervasive systems. ACM Trans Auton Adapt Syst 7(1): Article 12.
156. Weyns D, Schmerl B, Grassi V, Malek S, Mirandola R, Prehofer C, Wuttke J, Andersson J, Giese H, Göschka K (2013) On patterns for decentralized control in self-adaptive systems. In: de Lemos R et al. (ed) Software Engineering for Self-Adaptive Systems II, Springer, LNCS 7475: 76–107.
157. Mamei M & Zambonelli F (2009) Programming pervasive and mobile computing applications: The TOTA approach. ACM Trans Softw Eng Methodol 18(4): Article 15.
158. Castelli G, Mamei M, Rosi A & Zambonelli F (2015) Engineering Pervasive Service Ecosystems: The SAPERE Approach. ACM Trans Auton Adapt Syst 10(1), Article 1.
159. Riboni D & Bettini C (2011) COSAR: hybrid reasoning for context-aware activity recognition. Pers Ubiquit Comput 15: 271–289.
160. Cook DJ, Crandall A, Singla G, & Thomas B (2010) Detection of social interaction in smart spaces. Cybern Syst 41(2): 90–104.
161. Chen J, Dourish P, Hayes GR & Mazmanian M (2014) From interaction to performance with public displays. Pers Ubiquit Comput 18: 1617–1629.
162. Mäkitalo N, Pääkkö J, Raatikainen M, Myllärniemi V, Aaltonen T, Leppänen T, Männistö T & Mikkonen T (2012) Social Devices: collaborative co-located interactions in a mobile cloud. Proc International Conference on Mobile and Ubiquitous Multimedia (MUM'12): Article 10.

163. Campbell A, Collier R, Dragone M, Görgü L, Holz T, O'Grady MJ, O'Hare GMP, Sassu A & Stafford J (2012) Facilitating ubiquitous interaction using intelligent agents. Human-Computer Interaction: The Agency Perspective, Studies in Computational Intelligence 396:303–326.
164. Margara A, Urbani J, van Harmelen F & Bal H (2014) Streaming the Web: Reasoning over dynamic data. Web Semantics: Science, Services and Agents on the World Wide Web 25:24–44.
165. Maarala AI, Su X & Riekki J (2014) Semantic data provisioning and reasoning for the Internet of Things. Proc International Conference on the Internet of Things (IOT): 67–72.
166. Araujo R, Igreja A, de Castro R & Araujo RE (2012) Driving coach: A smartphone application to evaluate driving efficient patterns. Proc IEEE Intelligent Vehicles Symposium (IV): 1005–1010.
167. Wu C, Zhao G & Ou B (2011) A fuel economy optimization system with applications in vehicles with human drivers and autonomous vehicles. Transportation Research Part D: Transport and Environment 16 (7): 515–524.
168. Rogers Y, Connelly K, Tedesco L, Hazlewood W, Kurtz A, Hall RE, Hursey J & Toscos T (2007) Why it's worth the hassle: the value of in-situ studies when designing Ubicomp. In: Krumm J et al. (ed) UbiComp2007: Ubiquitous Computing, Springer, LNCS 4717: 336–353.
169. Ranganathan A, Al-Muhtadi J, Biehl J, Ziebart B, Campbell RH & Bailey B (2005) Towards a pervasive computing benchmark. Proc IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005): 194–198.
170. Yang HI, Chen C, Abdulrazak B & Helal S (2010) A framework for evaluating pervasive systems. Int J of Pervasive Computing and Communications 6(4): 432–481.
171. Schaub F, Könings B & Weber M (2015) Context-adaptive privacy: leveraging context awareness to support privacy decision making. IEEE Pervasive Computing 14(1): 34–43.
172. Atzori L, Iera A & Morabito G (2010) The Internet of Things: A survey. Computer Networks 54(15): 2787–2805.
173. Li S, Xu LD, Zhao S (2015) The internet of things: a survey. Inf Syst Front 17(2):243–259.
174. Miorandi D, Sicari S, De Pellegrini F & Chlamtac I (2012) Internet of things: Vision, applications and research challenges. Ad Hoc Networks 10(7): 1497–1516.
175. Valle ED, Ceri S, van Harmelen F & Fensel D (2009) It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems 24(6): 83–89.
176. Dourish P & Bell G (2011) Divining a Digital Future: Mess and Mythology in Ubiquitous Computing, The MIT Press.

Original publications

I Gilman E, Sanchez I, Saloranta T & Riekki J (2010) Reasoning for smart space application: comparing three reasoning engines CLIPS, Jess and Win-Prolog. Proc IEEE International Conference on Computer and Information Technology. Bradford, UK: 1340–1345, DOI: 10.1109/CIT.2010.240.

II Gilman E, Su X, Davidyuk O, Zhou J & Riekki J (2011) Perception framework for supporting development of context-aware Web services. International Journal of Pervasive Computing and Communications, 7 (4): 339–364, DOI: 10.1108/17427371111189665.

III Davidyuk O, Gilman E, Sánchez I, Mäkipelto J, Pyykkönen M & Riekki J (2012) iCompose: context-aware physical user interface for application composition. Central European Journal of Computer Science, 1 (4): 442–465, DOI: 10.2478/s13537-011-0031-z.

IV Gilman E & Riekki J (2012) Is there meta-level in smart spaces? Proc IEEE International Conference on Pervasive Computing and Communications Workshop on Managing Ubiquitous Communications and Services, Lugano, Switzerland: 88–93, DOI: 10.1109/PerComW.2012.6197638.

V Gilman E, Davidyuk O, Su X & Riekki J (2013) Towards interactive smart spaces. Journal of Ambient Intelligence and Smart Environments, 5 (1): 5–22, DOI: 10.3233/AIS-120189.

VI Gilman E, Sánchez I, Cortés M & Riekki J (2015) Towards user support in ubiquitous learning systems. IEEE Transactions on Learning Technologies, 8 (1): 55–68, DOI: 10.1109/TLT.2014.2381467.

VII Gilman E, Keskinarkaus A, Tamminen S, Pirttikangas S, Röning J & Riekki J (2015) Personalised assistance for fuel-efficient driving. Journal of Transportation Research Part C: Emerging Technologies, 58 (D): 681–705, DOI: 10.1016/j.trc.2015.02.007.

Original publications are not included in the electronic version of the dissertation.

528. Mantere, Matti (2015) Network security monitoring and anomaly detection in industrial control system networks

529. Piri, Esa (2015) Improving heterogeneous wireless networking with cross-layer information services

530. Leppänen, Kimmo (2015) Sample preparation method and synchronized thermography to characterize uniformity of conductive thin films

531. Pouke, Matti (2015) Augmented virtuality : transforming real human activity into virtual environments

532. Leinonen, Mikko (2015) Finite element method and equivalent circuit based design of piezoelectric actuators and energy harvester dynamics

533. Leppäjärvi, Tiina (2015) Pervaporation of alcohol/water mixtures using ultra-thin zeolite membranes : membrane performance and modeling

534. Lin, Jhih-Fong (2015) Multi-dimensional carbonaceous composites for electrode applications

535. Goncalves, Jorge (2015) Situated crowdsourcing : feasibility, performance and behaviours

536. Herrera Castro, Daniel (2015) From images to point clouds : practical considerations for three-dimensional computer vision

537. Komulainen, Jukka (2015) Software-based countermeasures to 2D facial spoofing attacks

538. Pedone, Matteo (2015) Algebraic methods for constructing blur-invariant operators and their applications

539. Karhu, Mirjam (2015) Treatment and characterisation of oily wastewaters

540. Panula-Perälä, Johanna (2015) Development and application of enzymatic substrate feeding strategies for small-scale microbial cultivations : applied for *Escherichia coli*, *Pichia pastoris*, and *Lactobacillus salivarius* cultivations

541. Pennanen, Harri (2015) Coordinated beamforming in cellular and cognitive radio networks

542. Ferreira, Eija (2015) Model selection in time series machine learning applications

543. Lamminpää, Kaisa (2015) Formic acid catalysed xylose dehydration into furfural

544. Visanko, Miikka (2015) Functionalized nanocelluloses and their use in barrier and membrane thin films

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O. Box 8000 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

UNIVERSITY of OULU
OULUN YLIOPISTO

ISBN 978-952-62-0957-9 (Paperback)
ISBN 978-952-62-0958-6 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)