OULU 2012
C 420

ACTA

UNIVERSITATIS OULUENSIS

Oleg Davidyuk

AUTOMATED AND INTERACTIVE COMPOSITION OF UBIQUITOUS APPLICATIONS

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

ACTA UNIVERSITATIS OULUENSIS
C Technica 420

OLEG DAVIDYUK

AUTOMATED AND INTERACTIVE COMPOSITION OF UBIQUITOUS 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 Wetteri-sali (IT115), Linnanmaa, on 8 June 2012, at 12 noon

UNIVERSITY OF OULU, OULU 2012

Acta Univ. Oul. C 420, 2012

Supervised by
Professor Jukka Riekki

Reviewed by
Professor Aaron Quigley
Professor Tommi Mikkonen

ISBN 978-951-42-9837-0 (Paperback)
ISBN 978-951-42-9838-7 (PDF)

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

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2012

Davidyuk, Oleg, Automated and interactive composition of ubiquitous applications.
University of Oulu Graduate School; University of Oulu, Faculty of Technology, Department of Computer Science and Engineering; University of Oulu, Infotech Oulu, P.O. Box 4500, FI-90014 University of Oulu, Finland
Acta Univ. Oul. C 420, 2012
Oulu, Finland

Abstract

Application composition refers to the creation of applications by using Web Services and resources such as mobile devices, displays, and various augmented everyday objects as building blocks. This approach is especially useful for ubiquitous applications which focus on supporting users' needs and everyday activities. This thesis proposes to compose these applications by choosing the appropriate set of resources and services and their configuration as dictated by the users themselves, together with their needs and other contexts.

This thesis studies automated and interactive application composition. The first approach enables the system to act upon users' needs and intentions, while the second enables users themselves to specify their needs and achieve their goals. The research was carried out by designing, implementing and evaluating eight prototypes for automated and interactive application composition. The evaluation methods used included performance analysis and user experiments.

The main results include (1) the design and implementation of automated composition mechanisms which rely on evolutionary computing and genetic algorithms, (2) a detailed performance analysis of these mechanisms using synthesized datasets and in a real networking environment; (3) the design and implementation of interactive application composition prototypes which rely on graphical and physical (i.e. touch-based) user interfaces for mobile devices and utilize various contexts; (4) an evaluation of these prototypes in a series of user experiments. The evaluation studied the following issues: users' attitudes towards an automated composition system which makes decisions on their behalf; users' attitudes towards a context-aware composition interface; and the issue of balancing user control and system autonomy.

Keywords: application composition, interaction design, physical user interface design, ubiquitous computing

Davidyuk, Oleg, Automatisoitu ja vuorovaikutteinen jokapaikan tietotekniikan sovellusten kokoaminen.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Teknillinen tiedekunta, Tietotekniikan osasto; Oulun yliopisto, Infotech Oulu, PL 4500, 90014 Oulun yliopisto
Acta Univ. Oul. C 420, 2012
Oulu

Tiivistelmä

Sovellusten kokoamisella tarkoitetaan sovellusten luomista käyttäen rakennuselementteinä Web Services -ohjelmistojärjestelmää, sekä muita resursseja, kuten mobiililaitteita, näyttölaitteita ja monenlaisia lisävarusteltuja tavanomaisia esineitä. Tämä menetelmä on erityisen käyttökelpoinen sellaisten jokapaikan tietoteknisten sovellusten luomisessa, joiden tarkoitus on tukea käyttäjän tarpeita ja päivittäisiä toimia. Tutkimuksessa esitetään, että tällaisia sovelluksia voidaan koota valitsemalla sopivia palvelu- ja resurssipaketteja sekä niiden konfiguraation käyttäjien vaatimusten mukaan, ottaen huomioon heidän tarpeensa sekä muut kontekstit.

Tässä väitöskirjassa tutkitaan automatisoitua ja vuorovaikutteista sovellusten kokoamista. Automatisointi mahdollistaa sen, että järjestelmä toimii käyttäjän tarpeiden ja päämäärien mukaisesti, kun taas vuorovaikutteisuuden ansiosta käyttäjä voi määritellä tarpeensa ja pääsee näin haluamaansa lopputulokseen. Tutkimus toteutettiin suunnittelemalla, toteuttamalla ja arvioimalla kahdeksan automatisoidun ja vuorovaikutteisen sovelluksen kokoonpanon prototyyppiä. Arviointimetodeina käytettiin suorituskykyanalyysiä ja käyttäjäkokeita.

Tutkimuksen tärkeimpiä tuloksia ovat (1) evoluutiolaskentaan ja geneettisiin algoritmeihin perustuvien automatisoitujen kokoamismekanismien suunnittelu ja toteutus; (2) näiden mekanismien yksityiskohtainen suorituskykyanalyysi syntetisoitujen tietoaineistojen ja todellisten verkkoympäristöjen avulla; (3) graafisille ja fyysisille (l. kosketus) mobiililaitteiden käyttöliittymille perustuvien ja monenlaisia konteksteja hyödyntävien vuorovaikutteisten kokoamissovellusten prototyyppien suunnittelu ja toteutus; (4) prototyyppien arviointi erilaisin käyttäjäkokein. Arvioinnissa tutkittiin käyttäjien asenteita käyttäjän puolesta päätöksiä tekevää automatisoitua kokoamisjärjestelmää kohtaan, käyttäjien asenteita kontekstitietoista kokoamiskäyttöliittymää kohtaan, sekä käyttäjälle jäävän kontrollin ja järjestelmän autonomian välistä tasapainoa.

Asiasanat: fyysisten käyttöliittymien suunnittelu, jokapaikan tietotekniikka, käyttäjävuorovaikutuksen suunnittelu, sovellusten kokoaminen

To my parents, Elena and Stanislav Davidyuk

Acknowledgements

First of all, I would like to express my sincere appreciation to my thesis advisor Prof. Jukka Riekki for supporting and guiding me patiently from the time when I joined MediaTeam Research Group as an MSc thesis student. Thank you for your kind support.

I would like to acknowledge all my colleagues who contributed to the development and implementation of the system prototypes and applications throughout this thesis project. In particular, I wish to express my gratitude to István Selek, Iván Sánchez, Ekaterina Gilman, Josu Ceberio, Jon Imanol Duran, Jussi Mäkipelto, Marta Cortés and Mikko Pyykkönen. Your efforts have been vital for the successful realization of all these prototypes.

I acknowledge Prof. Valerie Issarny and Dr. Nikolaos Georgantas for supervising and guiding my research during my visits to the ARLES project-team at INRIA.

Finally, I wish to express my thanks to Gordon Roberts and Richard James for their revision of the language.

My work on this thesis has been funded by GETA (the Finnish Graduate School in Electronics, Telecommunications and Automation), by the Academy of Finland and the National Technology Agency of Finland (TEKES), the Nokia Foundation, the Tauno Tönning Foundation, the Walter Ahlström Foundation, the Elisa HPY Research Foundation, the University of Oulu Scholarship Foundation and the Research and Training Foundation of TeliaSonera Finland Oyj. Their financial support is appreciated.

Abbreviations

AI	*Artificial Intelligence*
AIRES	*Automatic Integration of Reusable Embedded Software*
ANSO	*Autonomic Networks for Small Office/home office users*
BPEL	*Business Process Execution Language*
BRITE	*Boston University Representative Internet Topology Generator*
CADEAU	*Collecting and Delivering Multimedia Content in Ubiquitous Environments*
CANS	*Composable, Adaptive Network Services*
CAPNET	*Context-Aware Pervasive Networking*
COCOA	*Conversation-based Service Composition in Pervasive Computing Environments*
CC/PP	*Composite Capabilities/Preference Profile*
EA	*Evolutionary Algorithm*
GA	*Genetic Algorithm*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*
MEDUSA	*Middleware for End-User Composition of Ubiquitous Applications*
MGA	*Micro-Genetic Algorithm*
MUSIC	*Self-Adapting Applications for Mobile Users In Ubiquitous Computing Environments*
NFC	*Near Field Communication*
OWL-S	*Ontology Web Language for Services*
PalpCom	*Palpable Computing Project*
PC	*Personal Computer*
PICO	*Pervasive Information Community Organization*
PiP	*Pervasive Interactive Programming*
PSC	*Pervasive Service Computing*
QoS	*Quality of Service*
REACHeS	*Remotely Enabling and Controlling Heterogeneous Services*
RFID	*Radio-Frequency Identification*

SEA	*Straightforward Evolutionary Algorithm*
SOAP	*Simple Object Access Protocol*
STEER	*Semantic Task Execution Editor*
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WSBPEL	*Web Services Business Process Execution Language*
WSCL	*Web Services Conversation Language*
WSDL	*Web Service Definition Language*
XML	*Extensible Markup Language*

Glossary of terms

Application Composition is an approach to create ubiquitous applications by composing the appropriate set of services and their configuration as dictated by users, their needs, the situation and other contexts. The result is an application which may consist of one or multiple services

Automated Composition is an approach to construct ubiquitous applications using an automated composition mechanism

Composition Mechanism is an intelligent mechanism capable of acting and making decisions according to users' activities and needs. A composition mechanism may rely on some existing heuristic or planning algorithm

Composition Process is a continuous process which manages the life-cycle of applications from their initialization to the execution and termination

Context is an aggregation of relevant information such as a user's location, his/her preferences and the situation in the environment

Interactive Composition is an approach to construct applications by employing user control which users exercise through appropriate tools and user interfaces

Interaction Method is a user interface to control application composition that may optionally rely on some composition mechanism

Service is a set of capabilities of a physical resource (e.g., a computation, storage, or I/O device) that can be accessed through some software interface

Ubiquitous Application is an application which can be dynamically linked to multiple resources providing services and human-computer interaction capabilities for users

List of original publications

This dissertation is based on the following publications which are referred to in the text by their Roman numerals (I–VIII):

I Davidyuk O, Ceberio J & Riekki J (2007) An Algorithm for Task-based Application Composition. In the 11th IASTED International Conference on Software Engineering and Applications (SEA'07), Cambridge, MA, USA, November 2007: 465–472. URI: http://www.actapress.com/Abstract.aspx?paperId=32100.

II Davidyuk O, Selek I, Ceberio J & Riekki J (2007) Application of Micro-Genetic Algorithm for Task Based Computing. In the 1st International Conference on Intelligent Pervasive Computing (IPC'07), Jeju Island, Korea, October 2007: 140–145. DOI: 10.1109/IPC.2007.23.

III Davidyuk O, Selek I, Duran JI & Riekki J (2008) Algorithms for Composing Pervasive Applications. International Journal of Software Engineering and Its Applications 2(2): 71–94. URI: http://www.sersc.org/journals/IJSEIA/vol2_no2_2008/7.pdf.

IV Sánchez I, Davidyuk O & Riekki J (2009) Towards User-Oriented Application Composition. In the IEEE International Workshop on Pervasive Service Computing and Applications (PSCA'09), As part of the 4th International Conference on Frontier of Computer Science and Technology (FCST'09), Shanghai, China, December 2009: 698–704. DOI: 10.1109/FCST.2009.76.

V Davidyuk O, Sánchez I, Duran IJ & Riekki J (2008) Autonomic Composition of Ubiquitous Multimedia Applications in REACHES. In the 7th International ACM Conference on Mobile and Ubiquitous Multimedia (MUM'08), Umeå, Sweden, December 2008: 105–108. DOI: 10.1145/1543137.1543159.

VI Davidyuk O, Sánchez 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, IGI Global, Chapter 4: 74–103. DOI: 10.4018/978-1-60960-611-4.ch004.

VII Davidyuk O, Gilman K, Sánchez I, Mäkipelto J, Pyykkönen M & Riekki J (2011) *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.

VIII Davidyuk O, Georgantas N, Issarny V & Riekki J (2011) MEDUSA: A Middleware for End-User Composition of Ubiquitous Applications. In Mastrogiovanni F & Chong N-Y (eds) Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives, IGI Global, Chapter 11: 197–219. DOI: 10.4018/978-1-61692-857-5.ch011.

Contents

1 Introduction

1.1 Background

The origin of software composition research can be traced back to 1969, when Douglas McIlroy coined the term 'software component' and proposed the concept of *'coupling programs like a garden hose - screw in another segment when it becomes necessary'* in his landmark paper that motivated the whole software engineering discipline [1]. Nowadays, software composition is a widely used software engineering method for constructing and maintaining complex software structures using small software parts as building blocks. These parts are accessible runtime units of software, such as components, services, or some objects. Software composition research emphasizes the following three principles: i) reducing the complexity of the design and construction of software systems, ii) reusing software parts as much as possible, and, iii) combining these parts in many different ways to meet the requirements imposed, among others, by software developers, users and the environment.

Recently, software composition research has resurfaced with the emergence of ubiquitous computing. As the founder of this research field, Mark Weiser, described in his pioneering work [2]; ubiquitous environments are *future worlds* consisting of a number of different resources which integrate into human activities and support users' everyday lives. These environments enable ubiquitous applications which are dynamically linked to multiple resources providing services and human-computer interaction possibilities for users. In this context, a service can be defined as the capabilities of a resource (e.g., a computation, storage, or I/O device) that can be accessed through some software interface.

This thesis focuses on *application composition* to emphasize that the result of software composition has a user interface and is designed to help users to perform some task. This approach is specifically useful for creating ubiquitous applications as it proposes constructing applications by composing the appropriate set of resources and services and their configuration as dictated by the users, their needs, the situation and other contexts.

The application composition approach brings a number of benefits to ubiquitous environments. First, by composing services across multiple resources, a collection

of resources may be able to provide a seamless application functionality that would otherwise not be available. For example, a group of resource-limited devices can share their individual capabilities within the network and provide so-called virtual device functionality [3]. Other devices can combine and use these shared capabilities as part of a virtual device in order to extend their physical capabilities (e.g. lack of memory or sensors) or for the benefit of users. Second, application composition can be used to build multimodal user interfaces by combining and controlling inputs and outputs from multiple resources. For example, [4] uses this approach to enable rapid prototype development of applications integrating various interaction technologies (object identification, finger localization, tracking, etc). In addition, application composition allows ubiquitous applications to be developed that can dynamically adapt according to changes in the environment. For example, an application may need to recompose the set of services to achieve fault-tolerance [5] or to achieve late binding, which means making use of some services that are only available at runtime [6].

However, the main motivation for using application composition in ubiquitous computing is the requirement to support users' needs and activities. Users are essentially the focus of any ubiquitous system which always shapes the computation around what users need and how their needs can be supported. Application composition promises to meet users' needs by choosing the appropriate set of services and their configuration which create the application as required by the users. At a coarse grain, this requirement targets the issues related to the functionality of the application, quality of service (QoS), and context. The application functionality may serve multiple purposes for users including learning, home and office automation, task facilitation and pleasure, play and fun. Quality of service may serve as a measure of user satisfaction or as a quantitative characteristic that denotes performance and other application properties. Context refers to an aggregation of relevant information such as a user's location, his/her preferences and the situation in the environment [7]. The ability to support users' needs and activities is also reflected in application composition approaches used in ubiquitous computing. The research presented in is thesis is also motivated by this goal.

This research is multidisciplinary, and falls at the intersection of several established disciplines including software engineering, computer science, usability and human-computer interaction. The thesis combines methods and scientific practices of these disciplines in order to better suit the research problem and study application composition from different angles. Particularly, two topics are studied: *automated* and *interactive* application composition. Although the topics are closely related to each other, they are

addressed using different approaches. *Automated* application composition addresses computer science and software engineering, while *interactive* application composition targets human-computer interaction, usability and software engineering. The scientific practices and the methodology used are described further in Section 1.5.

1.2 Research scope, thesis statement and research objectives

As envisioned by Mark Weiser [2], a ubiquitous environment contains three essential elements: multiple resources providing various services, multiple applications utilizing these services, and multiple users interacting with these applications. The applications can be used by one or multiple users at a time, and each application can utilize one or many services simultaneously. Users interact with the applications by means of user interfaces. In contrast, the services interact (i.e. exchange information) with each other and with the applications by means of software interfaces.

A ubiquitous environment is naturally dynamic, as its users and some of the resources are mobile. The resources' properties (e.g. availability and load) are variable, as are the users' preferences and needs - which further complicates the matter. Evidently, application composition has to be dynamic in order to react to all these changes over time. Assuming that the environment provides a large number of resources, it might be appropriate to choose different sets of resources even to compose the same application at different times. Indeed, the characteristics of the resulting application mainly depend on the properties of the chosen resource set. Moreover, the application might need to be recomposed in order to react to changes that occur during application usage. In this setting, the high-level research problem for application composition can be specified as follows: How can one choose a set of resources, and their configuration, that supports a user's needs, and takes into account his/her preferences and the situation in the ubiquitous environment?

Theoretical aspects. This research problem requires the creation of a composition mechanism capable of finding the set of resources and their configuration that implement the application according to the multiple constraints and criteria specified by the users, software developers and characteristics of the environment. We consider that a composition mechanism has to be *efficient, robust, accurate and scalable*. An *efficient* mechanism consumes as little computational, memory and other resources as possible.

A *robust* mechanism guarantees to find a solution, if one exists, within a specified time period. An *accurate* composition mechanism finds solutions of high quality. A *scalable* composition mechanism demonstrates its suitability when operating with large problems.

A composition mechanism requires the creation of a model describing users, applications and resources in the environment. This model provides a foundation for the composition mechanism which uses the model's elements to build applications. We consider that the application composition model should essentially be *extensible, formal and generic*. An *extensible* model allows the easy addition of new concepts. A *formal* model enables state-of-the-art heuristics and optimization schemes to be applied. A *generic* model implies the ability to describe various application domains.

Practical aspects. As the requirement for any ubiquitous computing system is to be user-centric, the role of the user is also emphasized in this work. This research targets the following user-centric issues: the balance between the system's autonomy and user control in a composition mechanism, the user interface design, the utilization of context and the user experience. Moreover, in order to verify the system in a real world setting, this research introduces a further requirement: completeness, which is to say that the application composition system has to be a fully implemented solution. Therefore, the practical aspect of this research is centered around designing and implementing a system that uses the aforementioned model and composition mechanism, and resolves multiple research and engineering issues related to the implementation of application composition in a real-world setting.

These theoretical and practical aspects can be summarized in the thesis statement: *Providing a system which enables composing ubiquitous applications in a real-world setting.*

Hence, the first objective of this research is the construction of an application composition mechanism which has to be evaluated for efficiency, accuracy, robustness and scalability. The second objective is the construction of a fully functioning system prototype that provides appropriate user interfaces to enable user control for application composition, and the evaluation of this prototype in a series of user experiments.

1.3 Scientific contributions and the author's role

This thesis contains eight publications. The author of this thesis is the main author of all the publications, except for Publication IV. The prototypes of algorithms and systems described in these publications were built in close collaboration with a number of researchers. The scientific contribution of the individual articles, the roles of the author and the other researchers in each article are as follows.

Publication I introduces the graph-based model to represent formally application composition, and contributes to the design, implementation and performance analysis of the application composition mechanism based on the straightforward evolutionary algorithm (SEA). The idea for the paper was developed by the author of this thesis. The SEA algorithm is compared empirically with a greedy algorithm. The design of the evolutionary and greedy algorithms and models was undertaken by the author of the thesis. The implementation of the algorithms and the collection of data was performed by the author of this thesis together with Josu Ceberio. The analysis of the collected measurements was performed by the author of this thesis alone.

Publication II presents the design, implementation and performance analysis of the application composition mechanism which was based on the micro-genetic algorithm (MGA). The idea for the publication was developed by the author of this thesis. The MGA algorithm was compared empirically with the algorithm presented in Publication I. The task of designing the MGA was performed together by the author of this thesis and István Selek. The implementation of the algorithm and the data collection tasks were accomplished by Josu Ceberio. The task of analyzing the collected measurements was performed by the author of this thesis alone.

Publication III introduces the generic graph-based model for application composition, and sets out the design, implementation and performance analysis of two composition mechanisms. The idea for this publication was developed by the author of this thesis. The introduced mechanisms were the genetic and the evolutionary optimization algorithms which relied on the three-phase evaluation schema and used a customizable optimization criteria. The algorithms and the model were generic, i.e. were capable of describing different application domains. In addition, new concepts could be added to the model without redesigning the algorithms. The task of designing these algorithms was jointly accomplished by the author of the thesis and István Selek. The implementation of the algorithms was a shared task between István Selek and Jon Imanol Durán. The data was collected by Jon Imanol Durán. The author of this thesis

organized a series of experiments and was responsible for analyzing the collected data.

Publication IV describes the detailed performance analysis of the application composition mechanisms (presented in Publication III) working in two setups: as a standalone tool on synthesized datasets, and with a multimedia application when integrated as part of the REACHeS infrastructure. The idea for the paper was created together by the author of this thesis and Iván Sánchez. The solution, which integrates the application allocation with the REACHeS infrastructure, was designed together by the author of this thesis and Iván Sánchez. The solution was implemented by Jon Imanol Durán. The thesis author participated in collecting the measurements and analyzing the bottlenecks introduced by integrating the algorithm with the REACHeS infrastructure.

Publication V presents the system prototype which uses the genetic algorithm (presented in Publication III) to compose a multimedia application. The prototype and the application were fully implemented and were evaluated in a user experience study which involved 10 test subjects. This publication reports the study of user attitudes to an automated composition system making decisions on the user's behalf. The concept of the prototype was developed jointly by the author of this thesis, Iván Sánchez and Jon Imanol Durán. The prototype was implemented by Iván Sánchez and Jon Imanol Durán. The thesis author organized the user evaluation experiment, collected data through interviews and questionnaires, and analyzed and interpreted this data. The idea for this publication was developed by the author of this thesis.

Publication VI introduces an architecture, a system prototype and three interaction methods for application composition. Two of these interaction methods used the application composition mechanism presented in Publication III. This publication reports the comparison of the interaction methods in a large-scale user experience study which involved 30 test subjects. This study identified the factors and the contexts which affected the decisions of users to rely on the interaction methods. The publication also presents the study of finding a balance between user control and autonomy for the application composition system, using a smart newspaper application as an example. The idea for the prototype and the publication was created by the author of this thesis and Iván Sánchez, working together. The author of this thesis participated in the design of the interaction methods. The system prototype and the demo application were implemented jointly by Jon Imanol Durán, Iván Sánchez and Marta Cortés. The user evaluation experiments were organized and conducted jointly by the author of this thesis and Iván Sánchez. The author of this thesis carried out alone the tasks of analyzing and interpreting the data collected during the user experiments.

Publication VII sets out the design, implementation and evaluation of a system prototype for context-aware application composition. The prototype was fully implemented and evaluated with a learning application in a user experience study, which involved 21 test subjects. This publication reports a study of how various context data can be utilized to compose applications. This study also analyzes user attitudes towards a context-aware composition interface. The idea for the publication, the prototype and the demo application were created by the author of this thesis, who later also designed the user interfaces of the prototype and the application. The graphical icon design for mobile devices, and the graphical design of elements of the desktop application were drawn up by Mikko Pyykkönen. The system prototype and the demo application were implemented by Jussi Mäkipelto. The reasoner component was designed and implemented by Ekaterina Gilman. Iván Sánchez designed the REACHeS infrastructure used in this prototype. The author of the thesis was responsible for organizing and conducting the user evaluation experiment, collecting data through interviews and questionnaires, and analyzing and interpreting the results.

Publication VIII reports the design and evaluation of the middleware for end-user application composition (MEDUSA). The publication presents a user experience and feasibility study of a physical interface for end-user application composition. The idea for the publication was developed by the author of this thesis. The author accomplished the work alone, from the concept to the design, and conducted the user evaluation experiment. Dr. Nikolaos Georgantas and Professors Valérie Issarny and Jukka Riekki contributed valuable ideas for the middeware and suggestions for improving the manuscript.

Professor Jukka Riekki supervised the research presented in all the above mentioned publications, and advised the author of this thesis regarding the analysis of results, drawing conclusions and outlines of these publications. Nikolaos Georgantas and Professor Valérie Issarny supervised the research work for Publication VIII during the author's research visit to INRIA Paris-Rocquencourt.

1.4 Research history

The research presented in this thesis was carried out in the Context-Aware Pervasive Networking (CAPNET), UbiLife and Pervasive Services Computing (PSC) projects between 2006 and 2011 at the University of Oulu. In addition, this research was partly accomplished during the author's two visits to the ARLES project-team at INRIA (the

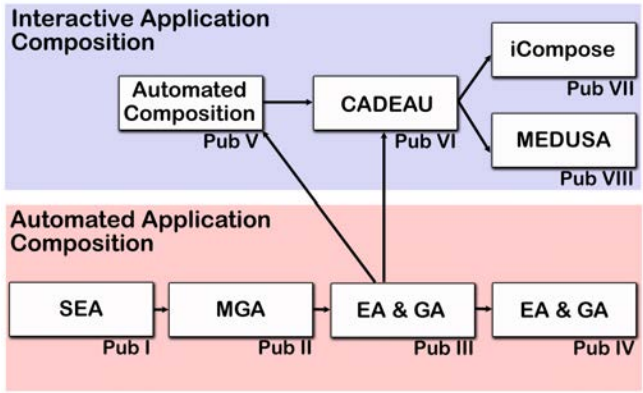

Fig 1. Evalution of the prototypes.

French National Institute for Research in Computer Science and Control) in the years 2008 and 2010.

This research started in the CAPNET project (2002-2007), which aimed to develop a component-based middleware for context-aware mobile applications [8]. In 2003, the author of this thesis joined the project, and took over the responsibility for developing the middleware component which a) controls the execution of applications and other components, and b) processes the requests for the components to gain access to each other. The idea for this thesis originated from this work; the idea was to support the composition of complex applications of components on physically distributed mobile devices. The challenge was to adapt the composed applications at runtime, in order to reflect changes in the mobile environment.

In 2006, the initial idea gave birth to the application allocation problem which described an application whose components need to be allocated onto the devices that constitute a ubiquitous environment. A solution to this application allocation problem (i.e. a mapping of the application components onto the device nodes) was subject to multiple constraints and optimization criteria. This problem was modeled using the concepts of graph theory. The initial prototype of the algorithm which solves the allocation problem was designed using the greedy optimization schema. Although capable of finding solutions on small-sized graphs, this algorithm had a rather simple design, and its performance characteristics were far from being excellent. The following short account illustrates this. The author was experimenting with the algorithm by running its instances on several departmental servers. Each experiment took from three to five days to complete. A few days after starting an experiment, some researchers

came to complain to the thesis author about the overloaded servers: the greedy algorithm was consuming almost all their computational capacity. Obviously, a better optimization schema had to be found.

During 2007, two alternative algorithm prototypes were introduced, one following the other. These algorithm prototypes were reported in Publications I and II, correspondingly (see Figure 1). Both operated with the graph models designed earlier for the greedy optimization schema. The first algorithm used the classical evolutionary schema, while the second one used the micro-genetic schema invented by Carlos Coello Coello [9]. However, despite the improved performance and smaller failure ratios, their computational loads and efficiency were only acceptable for small and medium-sized graph models. In addition, the graph models which we used to evaluate the algorithms were synthesized with our proprietary graph generator component. The value of our results would have been greater if we had used some well-known third-party graph generator for the experiments. This time the graph models had to be reconsidered, in addition to algorithm design.

In early 2008, the author thoroughly studied the related work in order to identify the common features of the formal models used in existing solutions. As a result, the author introduced a solution using customizable optimization criteria that was capable of working in different application domains (reported in Publication III). That is, new properties could be added to the model without redesigning the actual algorithm. The main goal, however, was the design of a new algorithm with better performance characteristics on large graph models. The new prototype introduced two algorithms: a modified genetic algorithm and an evolutionary one. These algorithms used a novel evaluation schema which allowed the time-to-convergence to be significantly improved. Both algorithms were tested using the widely known BRITE network emulator software tool [10] that randomly generates graphs and their properties. Finally, the performance characteristics of both algorithms were sufficient for them to be used in a real mobile environment.

Since 2008, the author of this thesis has changed the research focus from application composition mechanisms to system-wide issues, particularly feasibility and user experience. This shift can be also interpreted as moving towards the practical, applied side of this research problem. This shift was reflected not only in the significantly increased resource demands necessary for designing and implementing each subsequent prototype, but also in the overall design approach that changed to user-centered design. In particular, the success of the CADEAU and iCompose prototypes was mainly due to

this design methodology which involved the users in the early phases of the development process.

In spring 2008, we started working towards building a complete application composition system which could be demonstrated. The goal was to study user acceptance of the whole application composition concept, and the composition algorithms in particular. A simple multimedia player application was implemented for demonstration purposes. The algorithms were integrated as part of the REACHeS infrastructure [11] where they were responsible for the composition of the multimedia application upon users' requests. This prototype was automated, i.e. the users were supposed to start the composition and then only observe the result. However, we anticipated that users wanted to interact with the composition algorithms. For this purpose, we built an interactive poster which played the role of a remote control panel. Users triggered the composition by touching the NFC tags embedded in this poster with their mobile devices, which were equipped with NFC readers. The implementation was followed by two user evaluation experiments which studied user acceptance, feasibility and other issues. This prototype and the user evaluation experiments were described in Publications IV and V.

The results of these user experiments determined the most promising further research direction of this thesis project: user controlled and interactive application composition. Next, we designed three control interfaces for the application composition prototype (later, we called them *interaction methods*) and a new application scenario. This is how the CADEAU prototype was born. The idea behind this work was to compare the interaction methods and identify the situations in which the users preferred to use one method rather than another. Since we aimed to study the balance between user control and system autonomy in different contexts, these methods were designed to differ in how much the users were involved in the control of the application composition. CADEAU was designed and fully implemented in a very short time, during the summer 2008. However, the work on CADEAU was interrupted by the author's research visit to France and resumed one year later. The CADEAU prototype was evaluated in a large-scale user experience study which involved 30 participants. The results of this user study and the CADEAU prototype's design were presented in Publication VI. In addition, the author published a video deliverable [12] which demonstrated the setup of the user study, the designed interaction methods and the CADEAU application in action.

During 2008-2009, while visiting the ARLES research team at INRIA, the author designed the MEDUSA middleware, which focused on an interface for end-user (i.e. manual) application composition and the underlying software infrastructure. The

suggested design aimed to reuse the existing open-source solutions as much as possible. This work resulted in Publication VIII.

In 2010, the author refocused on developing the ideas which originated from the experiments with the CADEAU prototype. The goal was to create an interaction method that was capable of adapting itself to the user needs and the context. Such an interaction method promised a higher degree of user control and a more pleasurable user experience. As a result, we abandoned the application composition algorithms in favor of the context reasoner service [13] which took over the decision-making responsibility in our system. This was motivated by the fact that the context-based application composition offers greater flexibility in personalization than an optimization algorithm with customizable criteria. The iCompose interface that used the context reasoner service was designed and implemented as part of the prototype for application composition at the beginning of the year 2011. We also implemented the QuizBlasters learning application which we used together with the prototype during the user evaluation experiments. The work on the iCompose interface, the corresponding system prototype and the results of the user evaluation experiment were reported in Publication VII.

1.5 Research methodology

The research reported in this thesis followed constructive research principles [14, 15]. The research was performed through the development of real-life prototypes which were evaluated empirically. In general, the prototypes were developed iteratively, using Extreme programming methodology [16]. Each prototype underwent several development iterations, and selected evaluation methods were applied at each iteration. The number of iterations varied in each case and depended on the prototype's size. This means that larger system prototypes needed more than two consequent development iterations. This approach made it possible to continuously accommodate new ideas and features during the construction of the prototypes, while maintaining the focus of the implementations. The evaluation methods included simulations, evaluations in a real deployment environment, and user experience studies.

The algorithm prototypes reported in Publications I-IV were developed as a result of literature studies and a series of consultations with the experts in the field. After the development, the algorithms underwent a series of small tests in order to identify the values of the control parameters that resulted in the desired algorithm performance. These values were identified both empirically and based on consultations with experts.

Then, the author used these values in the experiments reported in publications. All algorithm prototypes were evaluated using the synthesized datasets. In addition, Publication IV reported the experiment which was organized with 30 real devices in the network of the University of Oulu.

The synthesized datasets were produced by either the graph generator component developed by the author (Publications I and II), or the BRITE network emulator software tool (Publications III and IV). The BRITE tool was chosen due to its status as the *de facto* standard in the research community.

The analysis of the algorithms was conducted using the metrics common for the research community, including performance, scalability, quality of solutions and algorithm failure rates. Due to the unique characteristics of the search problem (described in detail in Publication III), the algorithms were compared against either the greedy algorithm developed by the author (Publication I) or each other (Publications II, III and IV).

In contrast, the prototypes reported in Publications V-VII were implemented as fully functioning systems with demo applications. These prototypes were developed in several iterations, each followed by a user experience study. This made it possible to resolve the drawbacks and difficulties that our test users had stumbled upon during evaluations in the subsequent development iterations. The user experience studies were conducted to evaluate feasibility and interaction design and make an assessment of user acceptance of the developed prototypes.

The CADEAU prototype and the newsreader application (Publication VI) were evaluated in a large user study which involved 30 participants. The users were carefully selected to comprise three focus groups of 10 individuals each, representing IT experts, average technology users and conservative users. Each evaluation event lasted approximately an hour. At the beginning, all users were given a short introduction in which the functionality of the prototype was explained and demonstrated. Then, those users unfamiliar with the RFID technology utilized in the CADEAU interaction methods were given additional time to practice reading RFID tags. Then, users performed the tasks given by the organizers and provided their opinions about the prototype.

The iCompose prototype (Publication VII) was evaluated in a similar fashion. The user study involved 21 participants who were recruited according to their background and attitudes towards new technologies. The users formed two focus groups that consisted of advanced and novice users. The users arrived for the evaluation of the prototype in groups of two or three individuals. Each evaluation event took approximately 40 minutes.

The users were given several tasks to complete and were asked to finish evaluating the prototype when they felt they fully understood how the prototype works.

The organizers collected data using the notes taken during the evaluation, anonymous questionnaires and informal interviews. The questionnaires were used to compare the interaction methods, while the interviews collected feedback on the concept and the system prototype in general. The notes provided feedback about the behavior of users during the test. The anonymous questionnaires included multiple choice questions, questions with a 5-point Likert-scale (from strongly disagree to strongly agree), and open questions. After the users had filled the questionnaires, they were invited to discuss the difficulties they had experienced with the prototype, and also their ideas for future improvements. The organizers asked questions based on their earlier observations.

MEDUSA (reported in Publication VIII) underwent the conceptual design phase and was evaluated using the so-called low-fidelity prototyping approach. That is, the author designed a full-sized mockup of the MEDUSA user interface which was made of paper and carton. This interface was evaluated in a small-scale user experience study which involved two participants. The data was collected through observations during the experiment and also in an informal interview after the evaluation.

1.6 Structure of thesis

The rest of the thesis is organized as follows. The next chapter starts by discussing the principles of application composition and presents the concepts that form the foundation of our work, the user-centric application composition process. Then we move on to an overview of the field of automated application composition and discuss the state-of-the-art composition approaches in Chapter 3. Chapter 4 surveys the existing interactive application composition approaches, and discusses their user interfaces and the related underlying software infrastructures. Chapter 5 states the research contribution of this thesis in detail. Chapter 6 discusses the lessons learned, and highlights future research directions, while Chapter 7 concludes the thesis.

2 Composition of ubiquitous applications

2.1 Automated vs. interactive composition

Application composition approaches assume that it does not seem to be possible to design *a priori* the applications that satisfy all the possible needs of all categories of users in ubiquitous computing environments. As noted earlier, these environments are populated with services (i.e. ubiquitous devices that export their functionalities as services) which are capable of forming synergies in many different ways, and thus collectively achieving various functionalities which may serve different purposes in ubiquitous environments: pleasure, learning [17], play and fun, home and office automation [18–20], or task facilitation [21–23]. Some existing solutions focus on ensuring trustworthiness [24], reliability [25], fault-tolerance [5] and achieving the so-called late binding [6], which means making use of some services that are only available at runtime. Application composition can be used to facilitate multimodal user interfaces as proposed by Mungellini et al [4]. Their approach enables building multimodal user interfaces by integrating various interaction technologies (object identification, finger localization, tracking, etc). In addition, application composition can facilitate the so-called virtual device functionality [3, 26] by together composing shared functions of surrounding devices to overcome their limitations.

In this thesis, we define application composition as an approach to constructing the collective application functionality required to support the users' activities, needs and intentions. The result is an application which may consist of one or multiple services. The main guiding factor for application composition is the requirement to support the user's needs and activities, i.e. functional properties and user experience. Nevertheless, non-functional properties, such as QoS, security and reliability can also be used to guide application composition. For example, Lagesse et al [25] propose a composition mechanism which enhances reliability for composed applications by determining trustworthy compositions and hosts. Buford et al [24] target the issue of security and trust in application composition. Their approach is to apply the practices of digitally signed software to provide trust to single services. Ben Mabrouk et al [27] designed a QoS-aware composition mechanism which is capable of computing applications according to multiple QoS constraints. However, the author of this thesis

considers non-functional properties beyond scope of this research, and these issues will be not discussed further.

Generally, there are two approaches to application composition: the first uses automated intelligent mechanisms that are capable of acting and making decisions according to the users' activities and needs, while the second approach aims to enable users to achieve their goals and needs themselves, by providing appropriate tools to create and customize ubiquitous applications.

The first path is followed by *task-based computing* approaches which adhere to the minimum user-distraction principle in general [5, 21, 28]. Although users are expected to interact with applications, such tasks as application configuration, management and provisioning are performed by the system, i.e. with the goal to keep user involvement to a minimum. Ideally, users should not be concerned with these tasks at all, thus concentrating only on interacting with the application. These approaches assume that application developers (or, alternatively, users) create a template where they specify a set of service categories and their properties (i.e. functionalities) which are needed to achieve some user task. The system automatically chooses the appropriate services whose functionality matches the specified task template. Then, the chosen services are dynamically allocated for this task.

On the other hand, applications can be composed by users themselves, interactively. The composition approaches that follow this path provide user interfaces and various tools to enable users to compose applications [20, 29, 30]. In other words, interactive composition approaches employ user control which the users exercise through appropriate tools and user interfaces. The main difference between automated and interactive composition approaches lies in the fact that the first enables *the system* to act upon the users' needs and intents, while the second enables *users* to play an active role in defining the application functionalities that they need.

One specific subgroup of interactive composition approaches provides further flexibility by enabling *end-user composition* which allows *'...end-user creativity to emerge in a ubiquitous environment where people can create their own niche applications or adapt their ubiquitous surroundings'* [31]. End-user composition approaches support user control via an editor - a tool whose role is to allow users to create and edit composite applications and visualize the environment in the user's vicinity (i.e. the user's local environment). For example, editors are used in the following solutions for end-user application composition [18, 20, 32–34].

A brief chronology of most influential automated and interactive composition

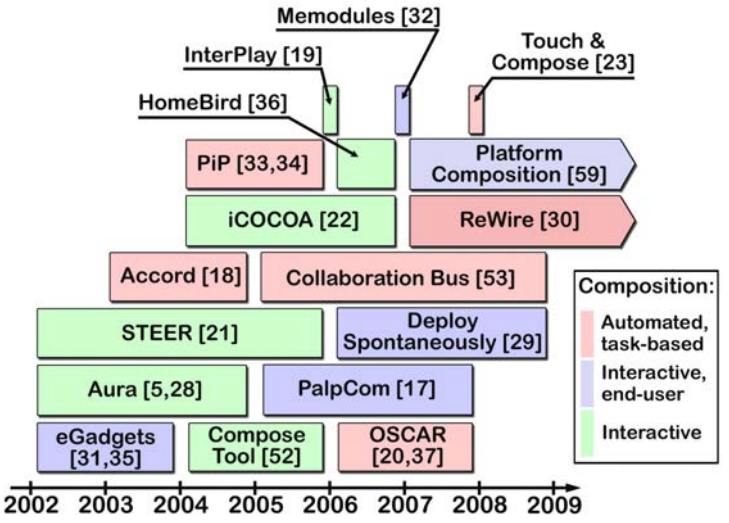

Fig 2. A brief chronology of influential composition approaches.

approaches is outlined in Figure 2. We discuss these approaches in more detail in Section 2.2 and Chapters 3 and 4 of this thesis.

To conclude, automated and interactive composition approaches differ in the degree of user involvement and control, although both focus on the same goal - supporting the users' activities and needs in ubiquitous environments. Automated composition approaches (presented mainly by task-based computing research) assume that user distraction has to be minimized and, therefore, they apply automated mechanisms to allow users to fully concentrate on interacting with their applications. On the other hand, interactive, and especially end-user composition solutions assume a greater degree of user control and, as a result, provide their users with greater flexibility and freedom. Solutions for interactive composition tend to enable users to express their needs, activities and various preferences themselves, while automated solutions rely on various context-aware mechanisms that deduce information about the users' activities.

2.2 The generic composition process

The generic composition process governs the life-cycle of applications from their initialization to the execution phase, as shown in Figure 3. The process starts with *the initialization phase*, which is a preparatory phase during which the basic elements for

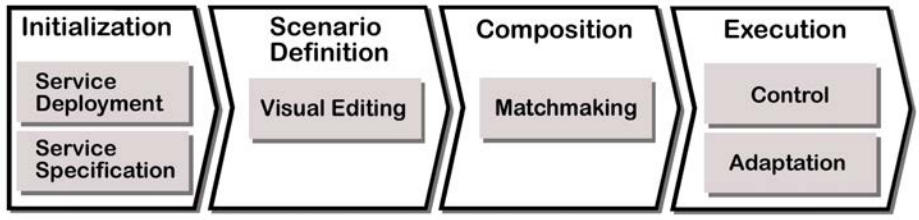

Fig 3. Workflow of the generic composition process.

the composition process (such as service specifications) are created and made available. During the second, *the scenario definition phase*, the template, i.e. the description of the future application is created. An application template usually consists of service specifications. Next, this template is transformed into a real application during *the composition phase*. The application that existed previously as a description is now being materialized into a real entity in the ubiquitous environment. Finally, *the execution phase* mainly governs the application usage and other real-time aspects such as, e.g. control and adaptation.

Initialization. In general, *initialization* implies creating the specification of services and making them available for application composition and usage. However, approaches differ in *how* this phase is carried out. For example, some application composition solutions assume that, in addition to service specifications, services have to be deployed and configured by users [29, 31, 35]. Service deployment is also emphasized in the Memodules project [32] which uses a Lay&Play tool to create digital counterparts (i.e. avatars) of ubiquitous artifacts, and adds these counterparts to a repository, so that they become available for the other phases of the composition process. In contrast, interactive composition [30, 33, 34] and task-based computing solutions [5, 21] assume that the initialization phase is limited only to creating service specifications, and adding them to a repository that supports various ways of searching (i.e. querying) on these specifications.

Scenario definition. During this phase, an application template or a scenario for the future application is created. Most automated composition approaches omit this phase, and assume that application templates are created at the same time as service specifications (i.e. during the initialization phase) [5, 22], although Aura [5, 28] and Interplay [19] do envisage a phase during which predefined application templates are customized to match user's needs. Unlike automated composition approaches, the scenario definition phase is essential for interactive (and especially end-user) application

composition solutions which assume that applications are created interactively using available scenario editing tools. Such an editor is usually based on a visual or a physical interface which captures the user's intent, and transforms it into an abstract (and formal) application template. The template is later materialized during the composition phase which results in a concrete application. However, some approaches use a different method which allows users to create and edit compositions that consist of concrete appliances (i.e. ubiquitous devices) available in the environment. These editors perform two functions simultaneously: template creation and application composition. Furthermore, the same editor can also be used to control and service the existing application in real-time, as designed, for instance, in the eGadgets project [31, 35].

Composition. This phase is central to the whole process, and usually involves a composition mechanism (or an algorithm) which takes an application description as input and produces (i.e. composes) a concrete application using the service instances that match the functionality specified in the description. A composition mechanism can be automated or interactive. Automated composition can be based on matching (e.g. syntactic or semantic matching) or knowledge inference algorithms (so called rule-based composition). Interactive composition supposes that the process of establishing application descriptions is governed manually by users through dedicated interface support [23, 36]. However, some end-user composition approaches employ physical interfaces which enable users to manipulate physical objects in the environment. This has been demonstrated, for example, in the PalpCom project [17]. Detailed analysis and discussion of automated and interactive composition mechanisms are presented in Chapters 3 and 4, respectively.

Execution. This phase governs application usage and various real-time aspects of application support, such as resource monitoring and adaptation control. For example, task-based computing approaches employ automated monitoring of services and dynamic adaptation in order to tackle service degradation issues. In particular, this issue is considered in the Aura project [5, 28] in a soft-failure scenario where one or multiple services become unavailable and, therefore, trigger the re-composition of the whole application. Automated re-composition may be necessary when a user's goals change dynamically [30]. Some solutions assume that the users themselves should be able to manually adapt (i.e. re-compose) the application, as it is not entirely possible to proactively anticipate changes in a user's goals. This feature is supported, for example, in the CADEAU and iCompose prototypes (Publications VI and VII). Real-time debugging of applications is another functionality that is necessary during the application execution

phase. This issue is, for instance, discussed by Mavrommati et al [31] who argued that real-time debugging and service browsing are essential tools for controlling composite applications. Automated monitoring and analysis of applications for diagnosing failures and supporting new services has been designed as part of the OSCAR project [37]. That project suggests sharing users' applications in a privacy-preserving manner, so that users are able to view and compare their own composite solutions, and thus choose the most appropriate ones that match their needs.

3 Techniques for automated application composition

The goal of a composition mechanism is to answer requests sent by users (or, alternatively, by the system) and compose applications that provide the requested functionality. Such a request describes the functional aspects of a composite application, for example, the required service interfaces, their public methods and input/output message parameters, service behavior and grounding. Some composition approaches assume that application descriptions (i.e. templates) are processed by the composition mechanism directly. Therefore, application descriptions are provided in a computer-readable form such as templates [5] or specification language queries [38]. Both examples specify functional requirements similarly. Another solution is to allow users to express what they need in a pseudo-natural language [39, 40]. Such language supports pseudo-English sentences that follow a predefined pattern which users need to fulfill in order to specify their requirements. For example, InterPlay [19] proposed a 'verb-subject-target' pattern to configure application tasks. Although this approach is straightforward for interactive composition systems, it requires translation of the sentences given in the pseudo-natural language to the specification language understood by the composition mechanism. Upon receiving an application description, the composition mechanism matches it with the service descriptions available in a service repository. The goal of this operation is to find the service combination that matches the application description. Matching (this operation is often called matchmaking) is performed according to the functional elements of the service descriptions which are usually the following:

– *Interface* is usually the most essential element of any service description. Interface describes the operations supported by the service. Generally a service interface can have multiple operations, each of which provides certain service functionality when executed.
– *Inputs and outputs* define the format of input and output values (or, sometimes, even messages) of the service or concrete service operations.
– *Behavior* describes the possible states of a service. Modeling of service behavior is needed when matching is performed on stateful services. A service is stateful if the execution of the service methods changes its internal state. Service behavior can be modeled using workflows [22], processes (BPEL) or pre- and postconditions. All

these examples are logical formulas that can describe a hierarchy of service states and their conditions. Exposing service behavior makes it possible to perform detailed (i.e., white-box-like) matching of service functionalities, although at the expense of greater complexity of the composition mechanism. On the other hand, stateless services do not require behavior modeling.

– *Grounding* provides information for invoking service methods. Grounding is usually a specification of access details, such as the concrete communication protocols supported by the service.

Matchmaking corresponds to the composition phase of the generic composition process (shown in Figure 3). As was mentioned earlier, this phase involves a composition mechanism which takes an application description as input and produces (i.e. composes) a concrete application using the service instances that match the functionality specified in the description. Next, we overview the three most prominent approaches for developing composition mechanisms. These are syntactic, semantic and policy-based composition.

3.1 Syntactic composition

Syntactic Composition is perhaps the oldest and the simplest composition mechanism used for automated systems. This approach annotates each service with a description containing a set of key-value attributes that specify service functionality. These attributes are characterized by numerical values, or by enumeration (i.e. ‘low’, ‘average’ or ‘high’). Service and application descriptions are represented as structured (i.e. with predefined syntax), or unstructured data, using a markup language such as XML. An example description of a composite application consisting of two services is shown in Figure 4. There, services are annotated with interface names, versions and availability properties.

There are several syntactic specification standards developed for automated composition. For example, Web Service Business Process Execution Language (WSBPEL) [41] and Web Service Conversation Language (WSCL) [42] concentrate on modeling Web Service behavior using processes (WSBPEL) or conversations and finite-state machines (WSCL). These specification standards can be used in conjunction with other standards, such as Web Services Description Language (WSDL) [43]. WSDL is an XML-based language that focuses on describing Web Service interfaces, and their methods and messages.

In order to compose an application using the syntactic approach, descriptions of

available services are compared against the application description which specifies the required functional properties. This operation is performed by a matching algorithm whose aim is to choose a service only if all of its description elements (e.g. interface names) match the elements specified in the application description.

Although the strongest point of the syntactic approach is simplicity and ease of implementation, this approach has a number of drawbacks related to extensibility and expressiveness which complicate its applicability. For example, changing the structure of service descriptions makes it necessary to redesign the matching algorithm and corresponding descriptions of all the services. This also means that if two services are described using different versions of the same specification language, these descriptions will not match. Another disadvantage of this approach is that it uses exact matching, i.e. the algorithm registers a match between a service description and an application template only when both descriptions contain exactly the same service properties. This means that descriptions of services and applications must *a priori* adhere to a common representation schema (or description language). Another disadvantage of XML is the lack of semantics in XML-based descriptions; that is, the meaning of descriptions may vary depending on the context of their usage [44].

3.2 Semantic composition

The semantic composition approach relies on service descriptions which adhere to a common ontology. An ontology is a domain-specific model describing types, properties and their relationships with explicitly defined and machine processable semantics [45]. For example, the Ontology Web Language for Services (OWL-S) [46] provides an

```
<task name="Messenger">
  <service interface="fi.oulu.context.LocationSensor" isAvailable="true">
    <settings key="service_url" val="http://10.10.1.16"/>
  </service>

  <service interface="fi.oulu.context.ContextBasedStorage" ver="2.0">
    <settings key="user_name" val="Davidyuk"/>
    <settings key="store_events" val="all">
  </service>

  <!-- ... -->

</task>
```

Fig 4. An example of a key-value based application description.

ontology for describing the Web Services domain. Using the semantic composition approach allows service descriptions to be combined with their meaning, which is impossible in syntactic composition approaches.

A typical semantic composition mechanism works as follows. Given a set of service descriptions, the mechanism selects the services that semantically match the application description. A match between a concept in a service and in an application description can be registered to a degree of similarity, despite syntactic differences between the concepts [44]. Another way for matching service and application descriptions is to use composability rules which determine, for example, whether two service candidates can be assembled in an application [47]. Such rules link service properties on several levels: the syntactic (e.g. service communication interoperability), the semantic (e.g. message parameters and service operation semantic), and the composition level (e.g. soundness of composition). A degree of semantic similarity can be also included in the set of composability rules.

Semantic composition has a number of advantages. First, it enables semantic heterogeneity among service and application descriptions provided by various parties, because it does not assume any agreement on description syntax. As a result, this approach allows service and application descriptions to be modified without changing either the composition mechanism (or matching algorithm) or service and application descriptions, which are annotated with descriptions of different versions. Second, the approach supports flexible matches, i.e. it detects whether two descriptions partially correspond to each other to some degree. Thus, it provides additional flexibility to the composition mechanism, but at the expense of an increase in false positives and lower efficiency.

The drawbacks of semantic composition include considerable development effort, and issues of interoperability between different ontologies. For example, semantic mismatches and conflict resolution between ontologies created by different vendor developers is an issue that has yet to be resolved.

3.3 Policy-based composition

Policy-based composition is another approach for developing a composition mechanism. Although this approach is more common in context-aware composition, it can also be used for user-centric composition [33, 48, 49]. The main principle of this approach is to create applications according to policies consisting of sets of event-condition-

action rules [33]. Actions are service methods that have to be executed when an event occurs, and a rule's conditions are met. Policies determine the behavior of the application and the environment (i.e., the user's personal space). For example, the policy-based approach in the Tara project [49] is used to compose services and manage their adaptive behavior. Tara's automated composition mechanism is based on an inference engine which performs matchmaking by reasoning about various functional properties of services. Policies in Tara comprise composition rules, utility rules (i.e. service interaction constraints) and behavior rules (i.e. behavior within the constraints that are provided for the composed application). Application adaptation is naturally specified as part of the behavior rules. All rules are classified as service level rules (i.e. that govern a single service) and high-level rules (i.e. that define constraints for the whole application). If policies are supplied by multiple parties (e.g. users, resources and execution environment) at the same time, the composition mechanism can use the hierarchical decision-making process as presented by Zhang et al [50]. In this process, all policies are divided into hierarchical groups depending on, for example, priorities of policies. A local decision-making (i.e. composition) process is executed for each group. Although policies are usually written by application designers, solutions that enable the users themselves to create their own application composition policies do exist [33].

Applying policy-based composition is straightforward for environments consisting of a large number of semantic services, where a high degree of automation is needed (such as in task-based computing). In spite of this, the approach suffers from the problem of policy conflicts. Conflicts may occur when the same event satisfies the conditions of two different rules, while each of these rules enforces actions conflicting with some other rule [51]. At the moment of writing this manuscript, the author is not aware of any existing policy-based composition approach which uses conflict resolution techniques.

3.4 Summary

Despite the obvious disadvantages of the syntactic approach, most of the related work surveyed utilizes it because it is straightforward and simple. Indeed, researchers prefer to adhere to some proprietary syntax for defining concepts rather than take into account possible incompatibility issues. In fact, these issues only occur if the application composition system has to be open and expandable.

Another tradeoff takes place between policy-based and semantic approaches. How-

ever, we consider this to be more of a design question for the following reason. The policy-based composition approach is straightforward for automated and interactive composition systems where events are crucial for maintaining system/application behavior. This approach is particularly promising for composition systems with adaptive and/or context-aware behavior. Indeed, policies are better suited for describing system behavior than semantic application descriptions, which are only able to capture structural (i.e. static) system properties. On the other hand, a policy-based composition approach might prove to be cumbersome and even redundant for static systems, where events only occur rarely.

Another advantage of the policy-based approach is its clear separation of concerns. Policies, in addition to composition rules, can describe system behavior and application logic. This makes it easier for application developers to create and modify composite applications by specifying policies.

Whereas this chapter surveys the application composition mechanisms that can be utilized to build an automated, as well as an interactive composition system, the next chapter focuses on user issues usually attributed to interactive composition systems.

4 Techniques for interactive application composition

In general, automated application composition approaches focus on minimizing the users' distraction in ubiquitous environments, and they assume that users should not be aware of how their tasks and applications are carried out. For this reason, user involvement in the application composition process is limited. Despite the high degree of automation of these approaches, their initialization requires the intervention of application designers who have to create service descriptions and application templates and configure the composition mechanism. This is done by setting up matching criteria or composition rules, if a policy-based composition approach is used. However, minimizing user involvement conceals the main disadvantage of the automated composition approach: although the system aims to support the user's needs, it does not guarantee that the resulting application will match the one anticipated by the users. Several researchers have taken steps towards resolving this issue and studied interactive application composition approaches. These solutions are discussed in this chapter.

In their pioneering work, Masuoka et al [21] presented STEER, a Web browser-based task composition tool which shows users what services are available in the current context and allows users to compose and execute the tasks. STEER is essentially a Webpage which consists of a set of drop-down menus, where each menu is associated with a service. By clicking on the drop-down menus, users can choose services and configure them using filenames or URLs. When all the services have been configured, users can execute the composed task from the task management screen. The undoubted advantage of STEER is its ability to create services that provide semantic versions of various digital objects (e.g. business cards, files and addresses). For this reason STEER has a drag-and-drop widget which 'consumes' digital objects dropped into it, and adds their semantic versions to the Webpage. This functionality is essential to allow the user to assign various objects with services' inputs and outputs. STEER also offers a user interface for task management, which allows users to relocate tasks by executing them using a local or a remote (networking) computer.

Similarly to STEER, Messer et al [19] presented the InterPlay prototype where the task composition screen allows users to choose and configure services available in their vicinity. However, instead of awkward drop-down menus, this approach uses visual

'verb-subject-target' constructions which resemble pseudo-English sentences. In order to execute a task, users choose first a 'verb' (i.e., a command 'play'), then a 'subject' (i.e., content) and, finally, a 'target' (i.e., a target device). Although this approach clearly offers a natural and an easy-to-learn user interface, it only supports pre-programmed (hardwired) tasks which have just two parameters to configure. In other words, users are not really able to compose tasks, but rather only execute the tasks that the designers have introduced into the system.

The task composition screen created by Sousa et al [5] resembles the idea for the STEER tool. However, Sousa et al's approach assumes that composition is triggered by dragging digital objects (e.g. a spreadsheet file) from a user's desktop onto the composition tool while the tool assigns a default command to these objects (e.g. 'edit spreadsheet'). Although this approach experiences the same problem as the Interplay prototype[19], (i.e. only pre-programmed tasks are supported) Sousa's tool is a unique solution that, in addition to automated application composition, focuses on capturing the users' task-related QoS preferences. In particular, Sousa et al's work allows tradeoffs between different QoS dimensions (e.g. latency and quality of content). Although default QoS dimension templates are provided for each task by the task designer, users are able to override default values through the QoS preferences interface. This interface is a pane with a set of QoS levels (from the lowest to the highest) where each QoS level corresponds to a value that can be specified by the user. QoS dimensions that are expressed by numerical values can be manipulated using slide bars, while QoS dimensions that are described by enumerations (e.g. 'high','medium' and 'low') can be manipulated using matrices of multi choice buttons. The QoS preference interface aims to capture how important are, or how much the user cares about, the quality dimensions of each service in the composite task.

Perhaps the most advanced editor among the interactive composition solutions is the Compose Tool developed by Wisner and Kalofonos [52]. Their solution is based on the idea that users should be able to select services for composition intuitively, while the system should automatically suggest possible interconnections between the services chosen by users. The editor, called the Compose Tool, has a GUI which consists of three panes, where the left pane shows a list of all known service types, the middle pane is an editing canvas which shows the services available in the environment, and the right pane shows the compositions proposed by the system. Users can drag-and-drop various service types to the editing canvas where they are immediately associated through the service discovery with the services available in the environment and are connected to

other services as proposed by the system. Users can choose from multiple proposals the one that best suits their needs. The Compose Tool uses the pipeline metaphor, where service compositions are depicted as directed graphs. Details of the pipeline metaphor are discussed later in this chapter.

4.1 Visual application composition tools

In contrast to automated application composition, interactive composition approaches exhibit a higher degree of user control, which is reflected in the number of visual composition editors presented in related work. A visual composition editor is a tool aiming to visualize services available for composition, provide an intuitive mechanism to create compositions and, optionally, display the state of the entire environment. Such a tool has a user interface which relies on a visual representation using graphics, animations or icons. A typical composition editor has at least two panels, a list of available service types or other elements (e.g. event templates) and an editing canvas (workspace). Users can drag and drop service icons or other visual elements into the editing canvas where these icons become links to concrete services in the environment. The workspace is an area where users arrange, connect and configure services. In general, interactive application composition approaches assume that the users should have the freedom to decide on what, how and when to compose applications. This is the main reason why editors for application composition are flexible and support a wide range of operations, though at the expense of greater complexity and longer learning curves. This chapter overviews visual composition tools, and classifies them according to the metaphor used.

The pipeline metaphor. Composite applications are represented as directed graphs, where nodes correspond to single services, while links (i.e., pipelines) are directed communication channels that connect two (or more) services together. Pipelines can be of a general type or can be assigned as a specific type (e.g. a video stream) which connects only certain services. Pipelines can be organized in complex structures using binary logic operators. For example, multiple pipelines can be connected with the AND operator. Also, pipelines can integrate processing filter elements which can be viewed of as conditional event generators. These filters trigger an event when data inside the pipeline matches some specified template such as keywords or numerical values.

The pipeline metaphor is a powerful visual programming paradigm which makes it

possible to model complex applications and their behavior, and this may involve networks of services interconnected by a complex eventing wire. Despite its expressiveness, editors using this metaphor have steep learning curves, which makes it difficult to apply this metaphor in user interfaces for inexperienced users. Therefore, it is either used in tools intended for expert users (e.g. CollaborationBus [53]), or requires a high degree of automation if the editing tool is meant for non-expert users. The latter is demonstrated in the Compose Tool prototype [52]. However, a simplified version of the metaphor has been used in OSCAR [20]. Unlike other approaches providing a work-pane where users can freely arrange and connect services, OSCAR allows users to operate only with pipeline templates. A pipeline template is an application consisting of two 'slots', a source and a destination, that have to be assigned with a service or some content. The editor uses a wizard-like approach to guide users through the steps that are necessary to populate a template. This means that OSCAR introduces the pipeline metaphor to non-expert users at the expense of limiting the range of application scenarios that can be supported.

The jigsaw puzzle metaphor. The principle of this metaphor is centered around a set of pieces which form a picture if correctly attached to each other. Players interact with the puzzle by fitting and attaching pieces which visually differ from each other in the number, shapes and locations of knobs and holes. These visual hints unambiguously indicate to players which pieces they may attach together. A jigsaw puzzle-based editor models each available service as piece of a puzzle. However, unlike in a real jigsaw puzzle where pieces always form the same picture, the composition editor assumes that composed applications may differ in structure and in the number of puzzle pieces (i.e., services) involved.

First, this metaphor promises a natural and an easy to understand interface, as indicated in [54]. Consequently, non-expert users can easily associate each piece of a puzzle with the visual service it represents. Moreover, the shapes of the pieces of the puzzle eliminate much of the mental effort necessary to understand the possible actions (i.e., which pieces go together). This makes it possible to use this metaphor to teach the principles and rules of a 'professional', and thus much more complex to learn composition tool, based on some other metaphor, such as, for example the pipeline metaphor [53]. Second, the puzzle metaphor naturally supports collaborative multi-user application composition, as two or more users can work on the same application simultaneously. Third, the puzzle metaphor can support complex structures (e.g.,

clusters of pieces of the same type) that act as single entities. Clusters facilitate end-user application design by decreasing complexity. These clusters may stay attached for a period of time while the user is composing an application. The downside of the jigsaw puzzle approach stems from the fact that the metaphor has constrained potential for expression: the pieces of the puzzle can have a limited number of interfaces (i.e. sides) thus narrowing the range of possible application scenarios.

One variation of this metaphor is the tile-based approach [55] which can be considered as a generalization of the jigsaw puzzle metaphor. The tile-based approach allows users to construct programs by manipulating and combining graphical building units (called tiles) with the mouse. Applications are assembled using tiles from some repository. Although each tile can be combined only with certain other tiles, the shapes of tiles are not physically limited, and can be connected with an arbitrary number of other tiles. A modification of this approach is presented by Warth et al [56]. Their system is intended for end-user programming and uses tile-based scripting. Their system combines a scripting interface (for designing programs textually), and a visual tile-based interface. The end users can switch freely back and forth between these two interfaces. Although promising, the tile scripting approach has not been yet used in the application composition domain.

Due to its simplicity and straightforwardness, the jigsaw puzzle metaphor has often been used in related work. For example, the Accord prototype [18, 57] includes a composition editor which is based on a tablet PC. Accord's editor makes it possible to interconnect simple sensors, web applications and various multimedia devices and services. A similar approach is used in the Action Builder editor developed in the Memodules project [32]. This approach is particularly interesting as their editor supports the composition of event-based applications. In this case, events are also modeled as pieces of a puzzle and can be composed together with certain services. Although, the Action Builder supports only one kind of event, this demonstrates that despite, the apparent simplicity of the jigsaw puzzle metaphor, it is capable of modeling a composite application that goes beyond trivial cases.

The join-the-dots metaphor. According to this metaphor, the editing canvas presents a set of individual devices that are available in the environment. Each device is shown as the center of a cluster, while the surrounding nodes represent services that a particular device offers. Users create compositions by drawing lines from one service to the desired destination device. Active compositions are shown as flows. In particular, this

metaphor has been applied in the editor of the Platform Composition prototype [58, 59]. A slightly different version of the same metaphor has been proposed in [31]. Their model, called the Plug-Synapse model, represents individual devices as circular nodes, and allows connections to be created by drawing lines between them. However, the plug-synapse model does not exhibit services offered by individual devices, thus making visual representation more compact. When drawing a line (i.e., making a composition) between two devices, users specify the composition on an 'association matrix' that visualizes a selection of concrete capabilities (i.e., services) offered by the devices chosen.

The advantages of this metaphor are the following. First, the metaphor uses a simple visual approach, as it shows the services (and devices) that are available only for composition. Second, the state of the entire environment is shown on the same screen. Indeed, it also shows the devices and services that are not used in any composition. On the other hand, this is a disadvantage because this metaphor is suitable for environments with only a few devices.

The application composition tools discussed in this chapter are closely related to visual programming languages. These languages rely on visual techniques throughout the programming process, and enable users to create, debug and execute programs by manipulating icons or other visual representations [60]. The programs are either compiled directly from their visual representations, or first translated into an underlying high-level textual language, and then compiled. Among successful examples of visual programming languages are the Google App Inventor (currently supported by the MIT Center for Mobile Learning)[1], and xUML [61]. The Google App Inventor is a visual programming tool which enables users unfamiliar with application programming to create applications for the Android mobile platform. xUML is a profile for Unified Modelling Language (UML) which allows users to graphically specify programs at a high level of abstraction independently from implementation-specific concerns.

4.2 Non-visual application composition tools

In addition to visual composition metaphors, some researchers advocate using non-visual composition tools, such as programming by example [62], and tangible (physical) interfaces [63]. The idea for the programming by example method can be explained

[1] http://appinventoredu.mit.edu/. Accessed 2012/04/23.

using the Pervasive interactive Programming (PiP) prototype [33, 34] that combines a visual editor and an AI mechanism, and works as follows. To begin with, users specify the devices they would like to combine together in the composition editor. Then, the users inform the editor that they are ready to program (i.e. compose) the application and they click on the start button. After that, the users interact with the real devices (e.g. physically, or by sending commands from a control UI) in the way they would normally use these devices in an application. At the same time, the AI mechanism observes the interaction and records the commands while trying to detect patterns which are later transformed into an application. This method has also been used to enable the construction and configuration of tiles, which play the role of building blocks in the Active Surfaces prototype [64] (this prototype is a part of the PalpCom project [17]). Each tile is an autonomous component which can be physically connected with other tiles. Tiles can communicate, and can recognize where they are placed. In Active Surfaces, expert users (i.e., therapists) create physical assemblies of tiles which follow a certain sequence or pattern. Later, tiles have to be placed in the correct pattern by end users (i.e. by patients) during rehabilitation sessions. The programming of tiles is performed in two steps. First, a therapist shows the correct pattern of tiles to the system, and then saves it using a special 'master tile' that communicates with all the tiles, and recognizes their location in relation to each other. A tangible user interface has been also used in FlowBlocks [65, 66] that enable children to learn abstract concepts related to counting, probability, looping and branching. FlowBlocks is a toy which consists of a set of tiles, each with embedded computation and magnetic connectors. Users create various structures (e.g. linear and circular chains) by connecting tiles together and then send a moving light signal through the tiles. Unlike Active Surfaces, FlowBlocks is oriented towards a larger user group, including children learning how to count, as well as high school or college students. Indeed, FlowBlocks supports the building of complex physical structures which can involve even elements of calculus and statistics.

The advantage of the programming by example method lies in its ability to hide most of the details of the underlying mechanisms from users in order to make it suitable for non-expert users. However, this method cannot be used in all cases of application composition. As pointed out by Mavrommati et al [67], *'there are cases when there cannot be a task example performed - i.e. because the application splits between different locations, different time periods or in situations that cannot be replicated.'* In conclusion, this method is not recommended for use as a stand-alone tool, although it can be used as a complimentary mechanism to any composition editor.

Non-visual composition tools can also be built using physical interfaces which rely on physical objects rather than on traditional I/O devices [63]. For example, a physical interface for application composition based on the jigsaw puzzle metaphor has been used in the MEDUSA prototype (Publication VIII). This solution represents service types with carton cards which have visual icons and RFID stickers glued to their surfaces. Cards can be arranged in various two-dimensional structures (sequences) which form an application, as needed by users. A mobile phone equipped with an RFID reader is used to capture (i.e. read) these structures by touching service cards in a certain order. A similar idea has been used in the CADEAU (Publication VII) and Touch&Compose prototypes [23], where the physical user interface consists of RFID tags and a mobile terminal. RFID tags are placed in the environment, and they indicate which services are available locally. Users compose applications by touching the corresponding tags with a mobile terminal.

4.3 Tools for runtime application composition control

Application composition approaches can also provide the means for user control at runtime. Although this phase can be controlled by automated mechanisms, various kinds of user input are required to deal with non-trivial situations. For example, Vanderhulst et al [30] and Hardian et al [68] argue that it would be unrealistic to delegate all decision-making to the system and expect it to figure out by itself what the user wants. In this case, user input is needed to deal with changes in user goals or context. For example, users might need to manually specify services (or devices) to be used with the application while it is being executed. This need is addressed in this thesis (see Publication VII): the author has proposed a physical interface to manually adapt composed applications by touching the desired devices in the users' local environment. In contrast, Vanderhulst et al [30] proposed a graphical user interface which is generated at runtime, and evolves with the environment. When a user triggers a reconfiguration of an application, the GUI loads the interactive wizard which provides access to application-specific features of the currently active application. However, most application composition approaches anticipate that users need to manage and service applications at runtime, and they provide basic support for starting and stopping application compositions. This is indeed the case in the following prototypes: CollaborationBus [53], PiP [33, 34], OSCAR [20], eGadgets [31], and ReWire [30]. A few approaches provide the application control using a physical (i.e. tangible) interface. For example, the Memodules prototype [32]

employs a tangible console for controlling multimedia applications. This works as follows. To begin with, users touch the console with various augmented objects in order to identify and trigger the corresponding application. After the console retrieves and starts the application, the users can control (i.e. interact with) the multimedia application using various touch sensors to play the next audio note, the previous one, to stop, and to adjust the volume. A similar idea was proposed by Kawsar et al [29] where the authors used RFID cards to control RFID-enabled augmented objects. In their prototype, each augmented object is supplied with a set of RFID cards, and an integrated RFID reader. Each card corresponds to a certain command (install/uninstall a service, start/stop an application). Users control augmented objects and applications by simply swiping a card - this action identifies the command (associated with the card) and the augmented object (associated with the RFID reader).

4.4 Summary

To sum up, non-visual tools are effective for only relatively simple applications and special cases when, for example, users need to compose applications by choosing physical devices. For this reason, non-visual tools are rarely used alone, but tend to be combined with an automated composition mechanism or a visual tool, as has been demonstrated in related work [20, 31].

In contrast, visual tools for application composition support a much wider range of applications and user groups. However, the suitability of a concrete visual editor for an application or a user group depends mainly on the metaphor used. The pipeline metaphor is a powerful and flexible visual tool which enables the composition of complex applications, albeit at the expense of steeper learning curves. Thus, this metaphor can be recommended for complex applications and expert users. The jigsaw puzzle metaphor, in contrast, offers a natural visual interface, and is thus easy to understand. However, the supported applications have rather simple structures. This metaphor is therefore more appropriate for simple applications and inexperienced users. The join-the-dots approach is a good alternative between the previous two metaphors. It offers an easy to understand representation, and application structures that go beyond trivial cases. This metaphor can be recommended for inexperienced users, especially when they need to be able to see the state of the entire environment.

User control in runtime application composition raises a number of open challenges which, so far, have not been fully addressed in the related work. For example, user

control in runtime application adaptation is limited in the solutions surveyed. This is possibly due to the fact that some composition approaches assume that user activities and needs are fixed and do not change after an application has been composed. This assumption is correct in situations where the composed applications have a relatively short lifespan, and focus only on users' immediate goals. On the other hand, recent research tends to promote continuous user interaction in ubiquitous environments [69], which results in a prolonged lifespan of applications, and gives rise to additional challenges related to application adaptivity. Hence, users need to have the possibility to control and adapt applications to accommodate changes in their goals and preferences at runtime. This requires developing various user interfaces (e.g. tangible, speech and gesture-based) to be used during the application execution phase in order to provide a better user experience.

5 Research and experiments

This thesis has two research themes: *automated* and *interactive* application composition. This chapter presents these research themes as follows: The first part of the chapter focuses on automated application composition, and presents the research methodology, implemented prototypes for automated application composition, and experiments with these prototypes. The second part of this chapter focuses on interactive application composition, and presents the research methodology, describes the implemented system prototypes for interactive application composition, the user experiments and the results of these evaluations. This structure reflects the research process of this thesis, which concentrated first on automated composition mechanisms and then shifted to interactive application composition.

5.1 Prototypes for automated application composition

The first research theme of this thesis focuses on various automated composition mechanisms for ubiquitous applications, and ways of improving the mechanisms' qualitative and quantitative characteristics. This theme corresponds to the first research objective of this thesis, which can be restated as *'the construction of an application composition mechanism which has to be evaluated for efficiency, accuracy, robustness and scalability'*. As part of the thesis project, the author developed four prototypes for automated application composition which were evaluated using synthesized datasets, as well as a real mobile application and network resources. Table 1 shows the relationships of the publications and the prototypes for automated application composition, along with their research goals and most distinguishing characteristics.

The prototypes of composition mechanisms presented in this section are implemented as standalone tools which are intended to be integrated into the system supporting automated composition of ubiquitous applications. The architecture of such a system is outlined in Publication III of this thesis. The core of the system consists of the application composer, the resource manager and the service discovery components. These core components realize the generic composition process (presented in Section 2.2) which governs the lifecycle of applications, monitors the utilization of the network resources and adapts the composed applications dynamically. The service discovery maintains the

Table 1. The algorithm prototypes developed in this thesis.

Prototype (Publication)	Design Goals	Design Description	Experiments
SEA (Publication I)	Solving the application allocation problem using the evolutionary computing approach.	Classical schema with all genetic operators implemented in Java.	Proprietary graph generator. The largest test setup had 11 application components and 26 networking nodes.
MGA (Publication II)	Improving the SEA performance while preserving the search quality.	Micro-Genetic Schema based on external memory & population reshuffling. Implemented in Java.	Proprietary graph generator. The largest test setup had 16 application components and 32 networking nodes.
EA&GA (Publication III)	Optimizing the algorithm according to the characteristics of the search space.	The EA uses the mutation operator only to mutate one individual. The GA uses the standard genetic schema. Both algorithms calculate fitness values using a three-phase evaluation schema. Both are implemented in C++.	BRITE Software Router generator. The largest test setup had 80 application components and 240 networking nodes.
EA (Publication IV)	EA feasibility studies in real world settings.	Integrated as a component of the REACHeS infrastructure. The EA is implemented in C++ and integrated using Java wrappers.	Tested using a real networking environment consisting of 30 resources and a media player application consisting of 3 components.

registry of network resources and performs a matchmaking functionality, which implies finding matches between the discovery requests and the service descriptions stored in the registry. The resource management component is responsible for monitoring the availability and the utilization of the network resources, as well as leasing them to application users. The application composer is the key component that uses the automated composition algorithms to optimally produce a concrete application using the service instances that match the functionality specified in the description.

The automated composition mechanisms, whose prototypes are presented in Publica-

tions I, II and III of this thesis, solve the so-called application allocation problem. This problem is based on an application and a platform model that are formally described as connected graphs. The application model captures the application's topology. Each node of this graph represents an application service. Communication between two services is modeled using graph links. The elements of the application model can have multiple properties which specify a non-functional service or link requirements, such as computational resources, up or downlink channel capacity and memory demand. The platform model represents the application execution environment. That is, the platform model's graph topology represents the network topology. It is assumed that each host is able to communicate with all other hosts in the network, therefore the platform is represented with a fully connected graph. Each host can allocate a group of application components, if the hosts' resource constraints are satisfied. These constraints determine, for example, the maximum memory, computation resource and other properties.

The number of properties affect the performance characteristics and the efficiency of the composition mechanism, as a greater number of properties cause the increased computational load and memory consumption of the mechanism. The prototypes of automated composition mechanisms (Publications I, II and III) differ in the number and the types of the supported properties. For example, the SEA and MGA prototypes (Publications I and II) supported 5 properties per model. In contrast, the EA & GA (Publication III) is a generic solution which supports multiple number of properties. This prototype was evaluated with 6 and 10 properties per model.

A composition mechanism uses the application and platform models as an input in order to produce a deployment plan. The deployment plan determines the optimal mapping of the application services onto the resources from the platform model. This mapping is subject to multiple constraints. The optimality of the plan is determined by an objective function. The SEA and MGA algorithms (Publications I and II) support the fixed objective function, while the EA & GA (Publication III) support the objective function which can accommodate new objectives without redesigning the algorithm's code.

The SEA, MGA and EA & GA prototypes (Publications I, II and III) were evaluated as standalone tools using synthesized datasets produced by the proprietary graph generator (the SEA and MGA prototypes), while the third-party network emulation software tool, BRITE, was used to evaluate the EA & GA prototype. In contrast, Publication IV reports the experiment which evaluated the EA algorithm when it was integrated into an existing infrastructure for application composition. This experiment

was performed in a real network with concrete devices.

Next, the prototypes and experiments are presented and discussed in detail.

5.1.1 The straightforward evolutionary algorithm (SEA)

The first algorithm was designed using the classical evolutionary computing schema described by Michalewicz and Fogel [70]. This algorithm is based on such evolutionary operators as tournament selection, one-point crossover and multipoint mutation. The platform model constraints are enforced using a penalty function and a fixed objective function is used to evaluate the potential individuals. This algorithm preserves population diversity by generating random individuals in the initializing phase and also by using a mutation operator after the population initialization. The algorithm was implemented as a Java standalone tool.

For benchmarking purposes, the author designed an algorithm that was based on a greedy allocation schema. This algorithm produced solutions by iteratively choosing an application component with the highest resource demands and allocating it to the node with the biggest resource capacity. The greedy algorithm handled the application and node constraints iteratively, one by one. Both the SEA and the greedy algorithm were evaluated for their performance and quality of search characteristics using synthesized datasets. To do so, we designed a proprietary solution for synthesizing application and platform models randomly.

The algorithms were compared empirically on relatively small datasets; in the largest case 10 application components were allocated onto 26 nodes. The small sizes of the experimental datasets were due to the fact that the performance of both the SEA and the greedy algorithm were poor and, therefore, unfit for runtime application composition (see Publication I for details). Moreover, the greedy algorithm was highly unstable, and suffered from frequent failures on larger datasets, in some cases finding solutions in only 5 per cent of the algorithm executions. Although the SEA outperformed the greedy algorithm in terms of robustness and quality of solutions, its computation times were of considerable magnitude (tens of minutes). As we established when analyzing the results, the reason for the time inefficiency lies in the SEA's diversity preserving mechanism which requires the algorithm to operate with large populations, usually of 150-200 individuals, to avoid premature convergence. Another drawback was the large number of algorithm parameters which we had to tune empirically through a series of experiments.

At the time of designing the SEA, we compared this algorithm with the standalone

solutions found in related work. The Sekitei [71] and Sekitei 2 [72] algorithms are AI-based solutions which aim to reduce the computational load of networking hosts, satisfy various QoS requirements, and improve the throughput using a dynamic service deployment approach. Although suitable for solving the application allocation problem, both algorithms focus exclusively on network applications where components produce or consume data streams. Similarly, CANS [73] focuses on network applications which can be composed as a direct sequence of components (i.e. sink-source chains). CANS optimizes various application-related metrics (e.g. overall throughput). AIRES [74] is an application allocation algorithm that is used to minimize hardware requirements during software design. However, AIRES computes application allocation statically, i.e. at the application design time.

Solutions such as DecAp [75], Hadas [76] and [77] introduce distributed algorithms. DecAp [75] focuses on improving the availability of distributed systems where network hosts are mobile and hence unreliable. The DecAp solution introduces the notion of partial visibility, i.e. it assumes that each host can access only a part of the overall network. DecAp allocates components according to the constraints imposed by partial visibility. Hadas [76] employs automated agents to negotiate about resource demands and leases for application composition. What makes Hadas unfit for application composition is the unpredictability of the negotiation process' duration, which depends entirely on agent behavior. A promising solution has been proposed by Andrzejak et al [77] which focuses on algorithms for automated systems management. However, to date, these algorithms have not been implemented.

As a conclusion, we found our approach to be promising and suitable for automated application composition, although, the algorithm designed had to be modified in order to improve the performance.

5.1.2 The micro-genetic algorithm (MGA)

The aim of this prototype was to address the drawbacks of the SEA, as well as to analyze the characteristics of the application allocation problem in more detail. In particular, we focused on improving the performance. This prototype used a micro-genetic schema [9] which is known for its low computational load. The main difference between this schema and the SEA algorithm lies in their diversity preserving methods. Unlike the SEA, which preserves diversity using only the mutation operator, the MGA relies on external memory and population reshuffling techniques. This simplifies the MGA's

design, and allows smaller populations with less than 10 individuals. The MGA is structured in two cycles, namely an external and an internal cycle. The external cycle controls the algorithm's memory, checks the stopping criteria, and restarts the internal cycle. The internal cycle resembles a classical evolutionary algorithm in miniature, as the MGA deals with a few individuals only. In addition, the MGA has two crossover schemes and two mutation operators to enable a faster convergence of the algorithm. Both algorithms, the MGA and the SEA, use the same penalty and objective functions. The MGA was implemented in Java as a standalone tool.

The MGA was compared empirically with the SEA algorithm on synthesized datasets. The largest dataset contained 16 application components and 32 platform nodes. Additional constraints were introduced in order to evaluate the performance of these algorithms. The results demonstrated that the prototype's goal had been achieved. The MGA's computational times were better, and its efficiency was the same as or better than that of the SEA, which is to say that the MGA found solutions of the same or higher quality. In terms of robustness, the MGA also demonstrated better results (see Publication II, Figure 8). The MGA was able to find solutions within a specified time period and failed less frequently than the SEA.

However, the experiments revealed that the search space in the application allocation problem does not contain information about the order in which the algorithm should sample the solutions. Moreover, the values of the objective function varied between neighboring solutions, and were independent of each other. The MGA's computational times were already in the magnitude of a few minutes. However, smaller computational times were needed for the algorithm to be used in a real-time application scenario.

When analyzing the MGA's design and performance, it should be mentioned that although the MGA's models are similar to those found in related work (e.g. [71, 72, 74]), additional constraints, called 'affinity constraints', were introduced. These constraints restricted the allocation of certain application components to certain hosts. At a coarse grain, this resembles the 'partial visibility' approach proposed in [75], but the latter approach uses these constraints in order to model the topography of mobile and adhoc networks. In the case of MGA, affinity constraints serve the need of application designers and also application users to adapt the allocation of application components according to their requirements. Other existing solutions, such as [5, 77, 78], were not capable of handling these kinds of constraints.

5.1.3 Modified evolutionary (EA) and genetic algorithms (GA)

The aim of this prototype was to achieve additional performance by exploiting features of the search space which had been discovered earlier. In addition, our goal was to make our algorithms generic, i.e. remove the need to tailor their models based on the type of application. Such algorithms can handle any number of properties as well as support new objectives without redesigning the code. Another goal was to integrate a third-party network emulation software tool to synthesize the datasets for experiments. The decision to abandon our proprietary synthesizer was due to the fact that our synthesizer produced datasets with slightly oscillating characteristics. Although all the parameters of the datasets were picked randomly, from specified value ranges, the resulting quality characteristics were oscillating around certain dataset sizes, as shown in Publication II (Figure 7). Hence, another tool had to be used.

In this prototype, two algorithms were proposed, the modified evolutionary algorithm and the genetic algorithm. The evolutionary algorithm had a very simple design, as no population was used. Indeed, the algorithm relied only on the mutation operator, and iteratively mutated one individual. However, the modified EA was expected to demonstrate higher performance at the expense of lower quality solutions. In contrast, the previously designed MGA and SEA were population-based algorithms. The second algorithm (modified GA) applied standard genetic operators, and cyclically evolved a population of individuals. These operators were: tournament selection, uniform crossover, multipoint mutation and elitism. The GA was expected to yield higher quality solutions at the expense of increasing computational times.

However, the main difference, compared with the earlier designed algorithms, lies in the evaluation schema for comparing the individuals. The new schema was designed in order to treat the application allocation problem treated as a hybrid search problem. That is, the new schema simultaneously combined the features of both constraint satisfaction and optimization problems. The schema worked as follows. First, the algorithms had to satisfy the platform constraints, until a valid solution was found (i.e., a solution which did not violate any constraint). After that, the algorithms had to optimize the objective function (subject to the stopping criteria). This approach made it possible for us to reduce the complexity of the search problem while supporting many optimization objectives without redesigning the code.

We implemented these two algorithms in C++ (as standalone command-line tools)

and used the BRITE network emulation software tool [10] to synthesize larger application and platform models. BRITE was chosen due to being widely accepted in the scientific community as a *de facto* standard tool at the time of these experiments. The two algorithms were empirically evaluated for performance and other characteristics while increasing the number of model properties and model sizes. Due to incompatibility issues, these algorithms were not compared with the earlier designed prototypes. First, the EA and the GA algorithms were tested while increasing the number of properties. Unlike the previous prototypes, these two algorithms were designed in C++.

The largest models we experimented with had 80 application components and 240 platform nodes. It should be noted that similar experiments in the related work operated with models of much smaller sizes (e.g., 20 components in the COCOA experiments [22]). The algorithms were also tested using models with 6 and 10 properties. The GA demonstrated slower performance in the cases used; however it produced solutions of higher quality than those found by the EA. In addition, the GA was more robust, and its failure rate was almost half that of the EA. Despite this fact, the EA's main advantage was its exceptionally short computational times (on average 17 times faster than the GA's). The conclusions of the experiments were the following. The utilization of the GA is better for finding initial application allocations (as it produces higher quality solutions), while the EA is more suitable when an application has to be reallocated (as the EA's search speed is faster). In addition, both algorithms suffered similar performance drops when the number of model properties was increased. The number of properties affected the complexity of the search problem, making it more and more complex, which consequently led to longer computational times. The design of these algorithms achieved its main goal: the algorithms' performance was acceptable for application allocation in real-time systems as their computation time was in the magnitude of milliseconds.

The main difference between these two algorithms and contemporary existing solutions (including our earlier designed prototypes) was mainly in the models used. Our algorithms operate with models in which additional properties and objectives can be added without redesigning the code. This means that the platform and the application models are not associated with a fixed number of properties.

5.1.4 Evaluation of the EA with the REACHeS infrastructure

Although the EA and GA algorithms demonstrated performance and quality characteristics that were acceptable for the application of these algorithms in a real-time system, this conclusion was based on the evaluation performed only with synthesized datasets, i.e. no experiments were performed in real world settings. Therefore, the next goal was to evaluate a chosen composition algorithm when integrated as part of an existing infrastructure for application composition. The EA was chosen for this evaluation due to its excellent performance and very low computational load.

The algorithm was integrated as part of the REACHeS infrastructure [11] which is responsible for managing resources and for supporting communication between resources, Web Services and mobile clients. REACHeS enables the utilization of a mobile terminal's user interface to control a wide range of ubiquitous applications, which can be composed dynamically of service components. In addition, a multimedia player application was implemented and integrated with REACHeS. This application supported various types of multimedia content rendered on wall displays remotely. This application consisted of three components: a remote control user interface (deployed onto a mobile terminal), a multimedia player user interface (deployed onto a PC) and a multimedia container service (deployed onto a Web service).

The application composition was triggered by end-users and was performed in two phases, as follows. In the first phase, REACHeS searches for the available nodes in the vicinity of the user (i.e. in the same physical location) and on the Internet. The result of this phase, the list of discovered nodes, is returned to the EA, which produces the application configuration during the second phase. This application configuration is the allocation of application components onto the discovered nodes according to multiple constraints and optimization criteria. Finally, REACHeS completes this application configuration by binding the required resources to the application. At this point, the user can start interacting with the application and REACHeS dispatches the user's commands, and other events sent between the application components. Details of how REACHeS coordinates and manages resources and applications can be found in [11].

The prototype of the application composition system was evaluated for performance and the latency caused by different services of the prototype. The goal of this evaluation was to analyze the factors that contributed to latency, while increasing the number of available resources. Another goal was to identify possible bottlenecks in the design. The

EA algorithm, implemented in C++, was integrated into REACHeS using Java wrappers. The network resources consisted of PCs connected to REACHeS. The latencies and the performance were measured during the presence of an increasing number of resources. The maximum number of simultaneously available resources was 30. As the experiment revealed, the algorithm execution latency dominated the other application composition processes, such as service discovery and various communication delays. The overall latency was below 100 ms, and this result was found acceptable according to the recommended response times for the UI design [79].

It should be noted that, unlike the contemporary frameworks and solutions focusing on task-based computing [5, 22, 28], this prototype was fully implemented, including middleware components, the user interface and the multimedia application. Details of the UI and the application were presented in Publication V of this thesis.

5.2 Prototypes for interactive application composition

The second research theme of this thesis is related to interaction methods for application composition which allow users to provide their preferences, and control various aspects of the application composition. This research theme corresponds to the objective of this thesis that can be restated as *'the construction of a fully functioning system prototype that provides appropriate user interfaces to enable user control for application composition, and the evaluation of this prototype in a series of user evaluation experiments'*. As part of the thesis project, the author developed several system prototypes for interactive application composition, which were evaluated in a series of user experiments. Table 2 shows the relationships of the publications and the prototypes for interactive application composition along with their research goals and the most distinguishing characteristics.

Table 2. The system prototypes developed in this thesis.

Prototype (Publication)	Design Goals	Design Description	Experiments
Automated Composition in REACHeS (Publication V)	Constructing a fully-implemented prototype which relies on a physical user interface and automated composition mechanisms.	The prototype was implemented in Java and integrated into REACHeS.	The user experiment with 10 participants demonstrated feasibility and usability of the prototype.
CADEAU (Publication VI)	Constructing a prototype which provides three user interaction methods for application composition.	The prototype is implemented in Java and C++, uses Web Services and is integrated into REACHeS Infrastructure.	The user experiment with 30 participants identified the factors and the contexts which affect the user's decision to rely on a certain interaction method.
iCompose (Publication VII)	Constructing a prototype which utilizes various user contexts for application composition.	Implemented in Java using the Ruby on Rails Framework, Web Services and REACHeS Infrastructure.	The user experiment with 21 participants analyzed users' attitudes towards utilization of context in application composition. The iCompose approach to deal with context was found acceptable.
MEDUSA (Publication VIII)	Design and evaluation of the middleware for end-user application composition.	Partially designed using available open-source technologies.	Evaluated in a user experiment with 3 users that demonstrated feasibility of the end-user application composition approach and suggested possible applications for the prototype.

The prototypes described in this section rely on a physical user interface to enable users to control the composition process which governs the life-cycle of applications (the process is described in detail in Section 2.2). While the automated composition

prototype, CADEAU and iCompose (Publications V-VII) focus on providing various interaction methods for application composition, the MEDUSA prototype (Publication VIII) implements the middleware for end-user application composition.

The automated composition prototype (Publication V) presents the fully-implemented system for application composition, and provides the automated interaction method, which relies on the EA algorithm (described in Section 5.1.3) for computing application configurations. In this prototype, the user control was limited to triggering the composition process with certain preferences. The CADEAU prototype (Publication VII) extended this interaction method and introduced two additional ones: the manual and the semi-automated method. The latter relied on the EA algorithm while the former allowed the users themselves to choose the necessary resources by touching them. These methods also relied on the physical user interface. However, each of these methods was advantageous in some situations, and disadvantageous in others. That is, none of the methods could be used as the universal method.

The iCompose prototype (Publication VII) presented the interaction method which tackles this issue. This interaction method utilizes multiple contexts and provides users with greater flexibility in terms of control. However, these advantages come at the price of longer learning curves and greater complexity for users.

All of the prototypes were evaluated in a series of user experiments in real settings. The automated composition prototype (Publication V) was evaluated with 10 users, while CADEAU (Publication VI) and iCompose (Publication VII) were evaluated with 30 and 21 users, respectively. In addition, the latter two prototypes were evaluated using the focus groups, which represented either experts and novice users (iCompose), or experts, average and novice users (CADEAU).

The MEDUSA prototype (Publication VIII) differs from the above described prototypes, as it presents the middleware for end-user application composition. Although MEDUSA does not directly continue the research on interaction methods started in Publications V and VII, this prototype is still closely related to interactive application composition. In particular, MEDUSA targets the issue of involving end-users during the scenario definition phase of the application composition process. For this purpose, MEDUSA provides the composition tool based on the physical user interface. This tool assists users in creating application scenarios which are later materialized during the composition phase. Users created applications by manipulating a set of service cards, where each card corresponded to an application service.

Fig 5. The control panel for choosing quality preference. (V, ©2008 Association for Computing Machinery, Inc. Reprinted by permission).

Next, we present the prototypes and discuss the user experiments and their results in detail.

5.2.1 The automated composition prototype

The goals of this work were to evaluate feasibility and user acceptance, as well as to study how user satisfaction and comfort levels are affected by the degree of autonomy of the system. The design of this prototype and its evaluation were essential in order to to gain a better understanding of user attitudes towards the autonomy of the application composition system and the implications of user interaction. Although the research focused clearly on user-oriented aspects of application composition, this prototype for application composition was designed in a similar fashion to the prototype for experiments with the EA algorithm (see Section 5.1.4). The application composition in this prototype was automated, and relied on the EA algorithm for computing application configurations. Users triggered the application composition by touching an NFC-tag on a control panel, where each tag corresponded to a certain quality preference (as shown in Figure 5).

Then, the prototype performed a service discovery and computed the application configuration according to the quality level chosen by the user. After that, the application was composed and started: the system automatically deployed the user interface components on the mobile terminal and on a certain wall display, so that the user

could immediately interact with the application. The prototype also supported runtime adaptation, i.e. users were able to reselect (and recompose) the application while watching the video. The prototype relied on three types of resources, wall displays (each controlled by a separate PC), media servers (which hosted the video content) and mobile phones (which were used as remote controls).

The prototype was evaluated with 10 users who represented pro-technology individuals. The setup of the experiment was the same as in the experiment reported in Section 5.1.4. The users were given the task of watching a video file while becoming accustomed with the interaction method and the prototype in general. The main findings of this study were the following. The prototype was regarded as feasible, and the users suggested various typical situations where they would need to use such a system. This helped us to identify two characteristics of the environments where the automated system for application composition will be useful: first, these environments are characterized by a large number of available resources, and second, these resources are not familiar to users, so that users experience problems when choosing suitable resources. The major finding was that the user's feeling of being in control was the most important factor. As most test users reported, the interaction method implemented with this prototype failed to provide adequate user control and definitely was not the one-size-fits-all solution to all possible applications. In spite of generally positive opinions about our prototype, users pointed out multiple drawbacks of limiting the system's applicability only to a few scenarios. We needed to design and study additional interaction methods to ensure wider applicability of our application composition approach.

At the time of this experiment, many system prototypes for automated application composition were available. However, only a few solutions were complete systems implementing user interfaces and tools, in addition to system components. Thus, we considered the following prototypes for comparison with our system: InterPlay [19], Sousa et al's prototype [5], the STEER tool [21], and the Compose Tool [52]. Interplay and Sousa et al's prototype use a similar approach for application composition, and they support only pre-programmed applications introduced to the system by application designers. Both systems support composition by choosing and configuring services available in a user's vicinity. However, Interplay uses a pseudo-natural language approach for composition, while Sousa et al's work relies on a GUI. Moreover, Sousa's approach focuses mainly on capturing application-related QoS preferences. In contrast, our approach relies on a physical interface to provide application-related preferences to the users. We consider a physical user interface to be a natural solution for this purpose,

as it has only a small learning overhead. Unlike Interplay or Sousa et al's work, we validated the feasibility and usability of our solution by organizing a user experiment.

Another two promising existing solutions are the STEER tool [21] and the Compose Tool [52]. Similarly to our approach, both solutions combine interactive user tools with automated composition mechanisms. This enables users to choose services for composition intuitively, while an automated composition mechanism creates interconnections between these services. In addition, the STEER tool features a set of widgets for introducing semantic versions of digital objects and services to the system.

5.2.2 CADEAU

The goal of this work was to study the balance between user control and system autonomy in application composition in various contexts, depending on the users' needs and experience with new technologies. To achieve this goal, we designed a prototype supporting three interaction methods which differed from each other in the degree of user involvement in the control of the application composition. These methods were offered in order to let the users choose the most suitable means of interaction according to their needs. The prototype was evaluated in a user experiment which aimed to address the question of system autonomy, i.e. the issues that users allow the system to take decisions on. The ultimate goal behind the design and evaluation of this prototype was to create an interaction method which can be tailored to the users' expertise and their willingness to delegate control to the system. Using such an interaction method, the users themselves could specify the tasks that they want to delegate to the system, and the tasks they prefer to control manually.

The CADEAU prototype supports the composition of applications of resources and Web Services. Similarly to the other prototypes for interactive application composition presented in this thesis, CADEAU has been built upon the REACHeS infrastructure [11], which provides the communication and Web Service remote control facilities, as well as performing service discovery and allocation of resources. CADEAU extended the basic composition and decision-making mechanism used in REACHeS and, additionally, provided three interaction methods for controlling the composition process. These methods were manual, semi-automated and automated. Figure 6 demonstrates how these methods are arranged according to the levels of user involvement and system autonomy that the methods provide. The idea behind providing three interaction methods at the same time was motivated by the users' need to switch from one interaction mode to

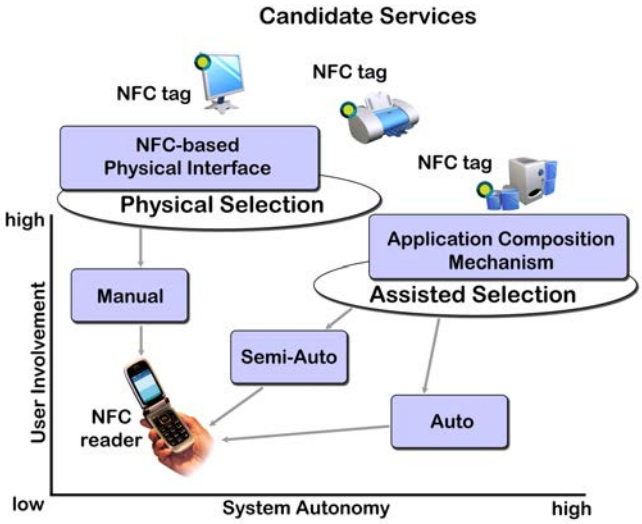

Fig 6. CADEAU interaction methods. (VI, ©2011 IGI Global. Reprinted by permission of the publisher).

another, as required by the situation in the environment, the application that is being composed and the users' individual requirements.

These interaction methods worked as follows. CADEAU's manual method addressed the users' wish to fully control application composition. This method allowed the users themselves to choose the required resources by touching the associated NFC tags with a mobile terminal. The semi-automated method allowed users and the system to control the application composition together. The method relied on the EA algorithm to generate dynamically a list of possible application configurations on the user's mobile terminal. Users first browsed and previewed these configurations, and then chose the one they preferred using the mobile terminal's UI. CADEAU's automated method was suitable for situations where users do not want to control application composition, and prefer to delegate this task to the system. When this method is used, the application is composed, configured and started automatically, without involving the users. For the purpose of demonstrating the capabilities of the CADEAU prototype, we designed a smart newspaper application which was also used for the user evaluation study. This application allows users to acquire various multimedia content by touching tags embedded in a newspaper. Then, users could browse and play the chosen content on an external display while controlling the playback using the mobile terminal as a

remote control.

The design of the prototype was followed by a major user evaluation involving 30 individuals with various backgrounds and previous experience with technologies. The main goal of this user study was to evaluate the tradeoffs between user control and the autonomy of the system dictated by the users' needs, situation and expertise. To achieve this goal, we compared CADEAU's interaction methods, and analyzed the various factors that contribute to the users' comfort level and feeling of being in control in different contexts. Each participant in the user study was asked to compose the CADEAU application using different interaction methods. The main findings of the study were the following. The manual interaction method was preferred when users wished to hide their multimedia content in the presence of other individuals. This method was also preferred when the environment and its resources were well known to the users (e.g. in one's office). The manual method also provided higher levels of user control and confidence. In general, users expressed similar preferences towards the semi-automated and the automated methods (details are presented in Publication VI). For example, these methods were preferred in situations where users were new to the environment or when the system's behavior was predictable. However, the semi-automated method was given preference if users wished to hide their intentions (e.g. when they aimed to discretely use public resources in the environment). As anticipated, the study confirmed that users' preferences towards these three methods varied, i.e. none of these interaction methods clearly dominated the others in any situation.

CADEAU's main research theme, the issue of balancing control between users and the system, has been addressed in several research prototypes. For example, Vastenburg et al. [80] developed a prototype to study and analyze a user willingness to delegate control to a proactive smart-home application. They developed a user interface for an atmosphere control application which had three modes of interactivity: manual, semi-automatic and automatic. However, their prototype was created using the 'wizard-of-oz' approach, i.e. the system's behavior and reactions were remotely activated by a human observer during their user study. Vanderhulst et al. [81] designed a meta user interface for runtime control of task-based applications through which users could manage their task execution, switch between tasks and perform 'personalized' service discovery. Hardian et al. [68] also studied the issue of balancing user control and autonomy. Their work suggested providing the user with control through explicit exposure of the system's logic and the context used for application adaptation through a user interface. However, the approaches presented by Vanderhulst et al and Hardian et al

focus exclusively on context-aware systems, whereas our solution allows application composition.

5.2.3 iCompose

The work on the iCompose prototype aimed at studying the context-aware composition of multiuser applications. Although the main goal was to develop and evaluate a user interface for controlling context-aware application composition and the runtime environment, this work also addressed the various technical feasibility issues of the approach. Indeed, the iCompose prototype was the most ambitious undertaking among the earlier prototypes, and its design required resolving multiple technical challenges such as multiuser interaction, distributed reasoning, context utilization and decision-making for composing and adapting applications.

Although the iCompose prototype was based on the REACHeS infrastructure [11], the prototype's design was a departure towards a generic architecture for context-aware composable applications. In addition, rather than using allocations algorithms, iCompose delegated decision-making to a distributed context reasoner. This change served two purposes. First, it separated context processing and management functionalities from application logic, and thus simplified the design of applications and the prototype components utilizing these functionalities. Second, supporting multiple composition processes simultaneously enabled an independent composition process for each application. This prototype featured the iCompose interface, which provided user control for the application composition process. This process runs in the background and triggers the composition, and adapts the applications according to context changes. The iCompose interface relied on a traditional GUI on mobile terminals, and on a physical user interface. The physical UI allowed users to interact with the environment and the system. The GUI (shown in Figure 7) visualized the events and the context related to the application composition, while the physical user interface offered the means to control the application composition. The iCompose interface incorporated a context reasoner which supported decision-making processes utilizing various contexts, and, optionally, implemented the application logic of some applications. This design was motivated by the following issues. First, the system's behavior has to be clear and unambiguous for users. Second, users should be given the authority to control the choices made (or suggested) by the system. Third, users should also be able to achieve their goals without the system's assistance. The prototype was accompanied with the

(a) (b) (c) (d)

Fig 7. The UI of the iCompose prototype: (a) the main menu, (b) event collector, (c) event collector during a quiz dialog, and (d) the Scorch remote controller screen. (VII, Reproduced from Central European Journal of Computer Science, 2011, Vol. 1, No. 4, Pages 442-465).

QuizBlasters application, both for demonstration as well as for user evaluation purposes. The application was a context-aware learning application which combined the elements of a treasure-hunt and a multiplayer action game called Scorch.

The fully implemented iCompose prototype was evaluated in a user study with the goal of assessing the usability, control and feasibility issues of the user interface for context-aware application composition. The user study featured 21 individuals representing novice and advanced technology users. All users were given the same task, which was to compose an instance of the QuizBlasters application while becoming familiar with the composition interface. The participants generally expressed positive opinions towards the prototype and the composition interface. The behavior of the system was reported as being 'as expected', and thus users raised only a few concerns related to context-aware features of the prototype. Most reported issues were linked to privacy and trust, which were within the scope of this prototype's design.

The existing composition approaches utilizing context and various context-aware mechanisms to carry out application composition and adaptation can generally be categorized into advanced context-aware systems and systems with basic support for context-awareness. The first ones are elaborate solutions utilizing a wide range of context types, and providing advanced functionalities such as context provisioning, context monitoring, service discovery and context reasoning. The second group focuses mainly on formal aspects of context-aware application composition, particularly on context modeling, context processing algorithms and specification of context-aware resources.

Example solutions with advanced context-aware support are presented in [82–85]. Bottaro et al [82] developed a system for automated policy-based application composition and adaptation which targets late-binding and user mobility issues. Late-binding and self-adaptation are the motivations behind MUSIC [84], a middleware for planning-based application composition. Focusing on utilizing the context provided by mobile devices, Hesselman et al [83] presented a composition approach that is tightly coupled with a service discovery and rule-based inference engine for interpreting context, and matching discovered devices. In contrast, the approach of Zhou et al [85] uses the rule-based reasoner to implement application logic and decision-making for application composition. Their solution targets the composition of pervasive Web Services. Examples of composition approaches focusing on formal aspects in context-aware composition are presented in [50, 86–89]. The solutions offered by Ben Mokhtar et al [86], Preuveneers & Berbers [87] and Zhang et al [50] address resource-awareness, late-binding and user mobility and perform dynamic application composition and adaptation using a semantic-based matching engine. Concentrating exclusively on user mobility, the PICO middleware [89], supports application composition and adaptation through semantic and syntactic matching. In contrast, the Diadalos architecture [88] targets personalization issues in application composition and includes multiple services for collecting and processing this kind of context.

The iCompose prototype can be categorized into the advanced context-aware systems group as it supports a wide range of context types, and provides context provisioning, reasoning and service discovery. Unlike related work solutions, iCompose focuses exclusively on interactive application composition, and features a context-aware physical user interface.

5.2.4 MEDUSA

Although this work continued the interactive composition research, the MEDUSA prototype represented a move towards end-user application composition. Indeed, the goal of this work was to enable the users themselves to create and customize simple applications from the available resources discovered in the users' vicinity. In contrast, our previously developed prototypes support only the composition of applications which were designed by application programmers. With a view to end-user application composition, the prototype focused on the following two issues: an application composer which enables end-user composition, and the interoperability of service discovery,

Fig 8. The MEDUSA composition tool: (a) A sticker with NFC tag, (b) an NFC-enabled mobile phone, and (c) example service cards. (VIII, ©2011 IGI Global. Reprinted by permission of the publisher).

service descriptions and service communication.

MEDUSA consisted of a composition tool for encoding user intentions into applications, a set of interfaces for controlling application composition, and the underlying middleware. The composition tool was based on the puzzle metaphor as follows. Each ubiquitous environment provides a set of cards which are associated with the service instances available in the environment. Each service is represented with a paper card with a graphical icon on one side, and with an NFC tag attached to the other side as shown in Figure 8. Users arrange these cards into different application structures which are then read by touching them with an NFC-enabled mobile phone. After specifying the application structure, users need to create control and data dependencies between the services using the mobile phone-based assistant. Then, the application has to be composed or, in other words, completed using concrete service instances. In other words, the services constituting the application have to be found and matched at runtime to the service instances available in the environment. This process is controlled through a set of user interfaces, including manual, semi-automated and automated interfaces. MEDUSA relies on exactly the same set of control interfaces as those which were developed for the CADEAU prototype. The last component of the MEDUSA prototype is the underlying middleware, which provides the necessary functionality for application composition, and enables interoperability between the basic elements of the environment, i.e. heterogeneous devices, platforms and networks. The MEDUSA middleware was designed using various open-source solutions.

We evaluated MEDUSA's end-user composition tool in a small-scale user experiment. The purpose of this experiment was to study the feasibility of the approach, and also

understand the types of application scenarios that can be created using MEDUSA's tangible composition tool. This brief experiment helped to identify the four most promising application domains (home, office, hospital and learning) where the composition tool was especially useful. Moreover, the experiment revealed that some sort of user motivation mechanism has to be designed in order to compensate for the learning efforts required of the users (see Publication VIII, page 13). Another result was the confirmation of our assumption that users need to be able to create and develop their own services, thus expanding their roles from end-users to service developers.

The end-user application composition presented in the MEDUSA prototype is related to research done in many different fields from composition middleware to graphical interfaces and editors for application composition. The prominent solutions featuring editors for end-user composition were proposed by Wisner and Kalofonos [52], Newman et al [20], Humble et al [18, 57], Mungellini et al [32] which we discussed in Section 4.1. However, none of these solutions offers a user interface for controlling application composition at runtime. Such support is required to ensure that users have a feeling of control over the environment, which is the major factor that affects user acceptability of application composition prototypes [90]. Also, unlike MEDUSA, the aforementioned solutions do not focus on supporting interoperability for service discovery, service descriptions and service communication.

5.3 Summary

This chapter discussed eight prototypes which were developed as a part of this thesis project. The prototypes were presented around two research themes: automated and interactive application composition. Likewise, we classify the main research results.

The automated application composition prototypes achieved the following results. These prototypes demonstrated that the performance of the developed composition mechanisms was acceptable for utilization in a dynamic environment. That is, the performance of the most recent prototype was in the magnitude of milliseconds. These results were supported by the evaluation of the final prototype (the EA & GA) using synthesized datasets, and in a real networking environment. In this sense, Publication IV produced the most interesting results, as it presented how the composition mechanism works when integrated with the system for application composition.

Another important result was achieving the flexibility of composition mechanisms, i.e. the ability to take into account new optimization objectives, and additional application

and platform model properties without redesigning the mechanism's code. This ability is necessary in order to ensure the applicability of the composition mechanisms in various application scenarios. However, this feature impacts the performance. As demonstrated in Publications III and IV, introducing additional constraints and model properties reduces the performance of the composition mechanism. Designers need to balance this trade-off when creating application composition systems.

The interactive application composition prototypes achieved the following results. The prototypes validated and demonstrated the feasibility of application composition. The user experiments (presented in Publications V-VII) confirmed the feasibility of the approach, and provided the author with lots of useful application scenarios and ways of utilizing these prototypes in the real world. However, the author had to spend a considerable amount of time on building the fully implemented prototypes and demo applications so that they could be evaluated.

In addition, the prototypes presented in Publications VI and VII studied the balance between user control and the system's autonomy in different contexts. These prototypes proposed several interaction methods which are based on a physical user interface. These methods were designed to differ in how much the users were involved in the control of application composition. The CADEAU prototype (Publication VI) presented the semi-automated and manual interaction methods, in addition to the automated method developed earlier (Publication V). The iCompose prototype introduced an interaction method which was capable of adapting itself to user needs and the context. As demonstrated during the experiments with iCompose prototype, this method showed a higher degree of user control and a more pleasurable user experience.

Another result was the middleware and the physical end-user composition tool developed as part of the MEDUSA prototype (Publication VIII). This work demonstrated that users should be involved also during the scenario definition phase of the application composition process. This allows the users themselves (rather than application designers in other prototypes) to outline applications as necessary. The user evaluation of the initial prototype supported the feasibility of this idea, though more extensive user evaluation is required to validate the prototype.

6 Discussion

In this thesis, we addressed the topic of composing ubiquitous applications. Focusing on the enabling infrastructure, we considered various composition mechanisms and interaction methods for controlling application composition. We presented the background to this thesis, ubiquitous computing, and an overview of application composition. With a view to providing a unified view of the thesis' topic, we considered two categories of approaches: automated and interactive application composition. We discussed the advantages and drawbacks of each of these categories. We presented the contribution of the thesis which is comprised of eight publications (explained in Section 1.3). We then analyzed them and compared with the related work.

6.1 Revisiting thesis statement and research objectives

Now, we may return to the thesis statement introduced in Section 1.2 and consider how the research, as presented in this thesis, addresses its objectives. We repeat here the thesis statement and the objectives for convenience: *'Providing a system which enables composing ubiquitous applications in a real-world setting'*. The first objective of this research was the construction of an application composition mechanism which has to be evaluated for efficiency, accuracy, robustness and scalability. The second objective was the construction of a fully functioning system prototype that provides appropriate interaction methods to enable user control over application composition, and an evaluation of this prototype in a series of user experiments.

The first objective was achieved as follows. The author developed, implemented and evaluated three consecutive prototypes of application composition mechanisms. These prototypes rely on the theory of evolutionary and generic computing to enable automated composition mechanisms. These prototypes were evaluated in terms of their efficiency, accuracy, robustness and scalability using synthesized network topologies and were also evaluated when working in a real-world network. The evaluation of the most recent prototype (reported in Publications III and IV) has demonstrated that the composition mechanism is suitable for runtime application composition and adaptation.

The second objective was achieved as follows. The author designed and implemented four prototypes which provided various interaction methods for controlling application

composition at runtime. In particular, the most recent prototype, iCompose (described in Publication VII), utilized multiple user interface modalities (a graphical and a physical user interface), and various contexts to enable user control for application composition. All of these prototypes were evaluated through a series of user experiments involving novice and expert users. Notably, the user experiment with the iCompose prototype confirmed its feasibility, and the usability of the interaction methods, as well as user acceptance.

6.2 Main contributions

This thesis contributes with i) algorithms for automated application composition, ii) user interfaces for interactive application composition, and iii) implementation of concrete applications and verification of the prototypes in a series of user experiments. Each of these is discussed in more detail below, followed by a consideration of open issues and future work.

Algorithms for automated application composition. We have designed and developed several prototypes of algorithms and models for application composition, which are reported in Publications I, II, III, and IV of this thesis. These algorithms apply principles of evolutionary and genetic computation to optimize the composition of applications using services and resources. The composition is subject to various constraints (availability and characteristics of resources, application components' resource demands) and user-specific criteria (such as minimization of communication overheads and maximization of QoS). The prototypes designed also include the graph-based models representing applications, their properties and the environment. The initial design of the algorithms and the models are reported in Publications I and II, correspondingly. The final prototype is described in Publication III. Unlike existing solutions at the time of publication, the final prototype has an extensive model which can be expanded to take into account additional properties, and constraints of applications and the environment. In addition, the model is generic and is suitable for describing multiple application domains. The algorithms have customizable optimization criteria, i.e. can be tailored according to the user's goals, without modifying the source code of the algorithms. This feature allows the algorithms and the model to be used in different application domains, such as entertainment, and home and office automation. The implementation of these algorithms and their experimental evaluation using synthesized datasets demonstrated their efficiency, accuracy, robustness and scalability. In particular,

Publication IV reports on the successful utilization of these algorithms and models in two real applications, and this demonstrates both their feasibility and effectiveness.

User interfaces for interactive application composition. Publications V-VIII discuss the user interfaces that we developed, together with the prototypes of application composition systems. Publications V-VII present interaction methods for controlling application composition at runtime, while Publication VIII introduces a design approach and user interfaces for end-user application composition. We identified the need to design these interaction methods during the work on the automated composition prototype (reported in Publication V). Although that prototype performed application composition automatically, the composition was nonetheless initiated by the users. Therefore, the automated prototype needed an interface through which users could trigger the composition and specify their preferences. Our solution was to design an NFC-based control panel; the action of touching a tag triggered the application composition with a certain set of preferences. Later, we designed manual, context-aware and semi-automated interaction methods, as presented in Publications VI-VII. Based on this work, we argue that there is no one-size-fits-all solution, but rather a set of methods should be offered, so that users are able to choose the most suitable method depending on their needs, the application, and the situation in the environment. At the time of writing this thesis, the author was unaware of any existing interaction methods for controlling application composition at runtime.

Implementation of concrete applications and verification of the prototypes through user experiments. The requirement to verify the application composition prototypes in a real world setting is specified in the thesis statement. In order to meet this requirement, the system prototypes underwent user evaluation, as presented in Publications V-VIII. Each prototype was verified and evaluated in a series of controlled user experiments which assessed various aspects, including general usability, user acceptance, feasibility, and user control. We decided which aspects had to be assessed according to the goals of each prototype. For example, user control was the most prominent feature to be analyzed for the system prototype which utilized automated mechanisms for application composition. To date, these two issues, completeness of implementation and user experience, are rarely addressed in existing prototypes for application composition. Instead of tackling system-wide issues, researchers prefer to focus on various challenges related to certain components of their prototypes, including automated decision-making and modeling application composition.

Recent work on these issues in application composition has produced interesting

research results. For example, Mukhtar et al [91] and Lahmar et al [92] have developed a solution for automated application composition and adaptation. Their composition mechanism considers various constraints, including user preferences, resource capabilities and the heterogeneity of communication protocols. Their models utilize extended CC/PP description profiles [93] and conditional-preference networks for compact representation of conditional and qualitative user preferences relations and resource capabilities. Moreover, their composition mechanisms support adaptation to certain contexts, such as changes in resource supply and user preferences.

6.3 Open issues

There are several issues which we only touched on or did not consider at all in this doctoral thesis. As such, this work does not directly address the issues of middleware interoperability for ubiquitous environments. Ben Mokhtar [94] specifies two kinds of middleware interoperability required for application composition: service discovery, and service communication protocol interoperability. The first kind of interoperability is required to overcome the semantic heterogeneity problem which occurs when several systems are using different description languages for information exchange [95]. Semantic heterogeneity entails incompatibility due to differences in the content, as well as in the way the same concepts are described. The second kind of interoperability is needed to overcome heterogeneity among various protocols used for service discovery and service-to-service communication.

We addressed the first kind of interoperability in the MEDUSA prototype (presented in Publication VIII) which uses ontologies to solve the semantic heterogeneity problem. Ontologies make it possible to model applications, services, resources, their properties and possible relationships using a common explicit specification of concepts (called 'upper ontology'), thus enabling communicating parties to search and match application concepts even if these concepts are described in different syntaxes. However, we resolved the second kind of interoperability only partially. For this purpose, we designed our prototypes for application composition (reported in Publications V-VII) using the REACHeS infrastructure [11] as a basis. REACHeS was originally designed as a solution for universal control in ubiquitous environments. REACHeS connects various ubiquitous resources and mobile terminals together using Internet browser technology. One could say that REACHeS enables interoperability for devices equipped with Internet access that are capable of communicating using some common protocol like HTTP

or SOAP. REACHeS connects and manages these devices, thus forming the resource pool that can be used to compose applications. Although the design and evaluation of the prototypes (reported in Publications V-VII) have demonstrated that this solution was sufficient for the needs of this thesis, additional work is required to achieve full interoperability of communication protocols. In general, this issue can be resolved by using various auxiliary components injected into communication paths, such as adapters and translators. Adapters implement the communication functionality of both senders and receivers, thus creating explicit mappings between various communication protocols. In contrast, translators implement only the communication functionality of senders. The former approach is presented, for example, in [96], while the latter is used in the ANSO architecture [97].

Two other issues that we did not address thoroughly are trust and privacy for ubiquitous applications. Trust is an essential feature, particularly for user-centric ubiquitous systems which are capable of making decisions on the users' behalf. We first recognized the importance of trust when we evaluated our automated prototype (Publication V), and later studied trust in the CADEAU prototype (Publication VI). We found that trust is a complex and multifaceted issue, interwoven with several factors: user expectations towards system behavior, user familiarity with the ubiquitous environment, the presence of other people in close proximity, and so on. For example, trust is high if the automated system behaves as anticipated by the users, otherwise it is low. This finding is also supported by Vastenburg et al [80], Hardian et al [68] and Dey at al [98]. We also noticed that trust is linked to psychological factors; the presence of other people (e.g. in a public space) directly influenced user confidence, so that users were afraid to try the system if it might cause some embarrassing situations (e.g. a break in silence). This is also related to user familiarity with the ubiquitous environment; the location and availability of resources, and the premises in general. We found that users were cautious and uncertain when introduced to a new and unknown place. Finally, trust is related to user confidence with various technologies, and their social acceptance. The acceptance of NFC, RFID and related ubiquitous technologies is addressed in [99, 100].

Privacy, like trust, directly affects the feasibility and user acceptance of ubiquitous products. Privacy in ubiquitous applications requires resolving multiple issues related to collecting, accessing and disseminating the users' personal data, as well as dealing with sensitive content or applications in public environments [101–103]. Although our prototypes did not focus on privacy, we revealed some issues during the user evaluation experiments reported in Publications V-VII. For example, our users were concerned

about being able to hide their intentions, like preparing to use certain resources with an application, in public environments or in close proximity to other individuals. Users voiced a similar concern when they wished to hide sensitive applications or content (e.g. when browsing a family photo album in a cafeteria). In these two cases, privacy was the main factor behind the users' choice of the manual control interface for application composition. In other words, our prototypes allowed users to enhance their privacy by choosing the control interface that suited their needs and the situation. This is also supported by Hardian et al [68], who suggested using explicit manual control for privacy-critical applications.

Still, the mainstream research dealing with privacy in ubiquitous applications focuses on the other aspect of privacy, i.e. collecting, accessing and handling the users' personal data [104–107]. Various protection strategies can be used to address these issues, including reducing the amount of private data being disseminated, reducing the amount of private date being collected and stored, privacy preserving data mining techniques, and controlling access to private data [108]. The first strategy in particular looks very promising. Indeed, the iCompose prototype (Publication VII) is based on a distributed reasoning mechanism which can be used to implement this kind of protection strategy. This reasoning mechanism allows private data to be processed in the users' own mobile terminals, thus reducing the need to disseminate private data to other computing nodes.

6.4 Future work

We have identified several promising research directions which could pave the way for future work. The first interesting research topic is related to incorporating various resource management mechanisms for ubiquitous environments. As we noticed when designing and evaluating the iCompose prototype (Publication VII), interaction flows involving multiple users simultaneously lead to situations where additional mechanisms are required to handle: i) concurrent access to resources and conflict resolution, ii) user mobility and handoffs, iii) control of QoS and degradation of resource properties and, iv) incorporating various policies for the optimization of resource usage. Tackling these issues increases the overall quality of service and the availability of resources [109]. These issues can be addressed by a resource management mechanism which is tightly coupled with a resource and service discovery engine, and processes the requests of both applications and users to gain access to certain resource instances in the smart environment. For example, the application composition system presented in this thesis

would benefit from incorporating an existing resource management solution, such as MetaGlue [110] or ScreenSpot [111].

Another promising research direction concerns supporting application designers. So far, our research has mainly emphasized support for end-users, and particularly for run-time composition, adaptation and control. However, providing application designers with adequate tools and design methodology for fast-prototyping ubiquitous applications is essential for wide-scale adoption of the application composition concept. Such support can be offered in the form of a set of libraries or even as a complete application framework. Such a framework would provide tools for specifying a composite application, creating and editing rules that define the conditions for triggering composition and adaptation. Moreover, such a framework would simplify the utilization of new types of context and additional interaction modalities in composite applications. This framework could be based on existing solutions for context management and multimodal interface design, such as the Perception Framework [13], the solution introduced by Cadenas et al [112], or the OpenInterface Framework [113]. At a higher level, Sousa et al [114] present a design methodology which focuses on creating ubiquitous applications using application composition.

7 Conclusions

This thesis shows that ubiquitous applications can be created by using Web Services and different resources as building blocks. This approach is referred to as application composition. It is argued that this approach is better suited for ubiquitous applications than the traditional design approach. Indeed, application composition emphasizes i) combining resources and Web Services in many different ways to meet the requirements imposed by application designers, users and the environment, ii) reusing resources and Web Services as much as possible, and iii) reducing the complexity and the cost of application design and maintenance. However, the main motivation behind the research presented in this thesis is the requirement to support the user's needs and everyday activities. Application composition promises to support this vision by choosing the appropriate set of resources and Web Services and their configuration as dictated by the users themselves, together with their needs and other contexts.

The application composition approaches studied in this thesis are classified into automated and interactive composition. Although these approaches focus on the same goal, i.e. supporting the user's needs and activities, they differ in how this goal is achieved. Automated application composition adheres to the 'minimum user distraction principle', so that users can concentrate only on interacting with the application. Application configuration, management and provisioning tasks are performed by the system without involving the users. This is achieved by using automated mechanisms which make decisions based on the information available about the users' activities and needs. Application developers create applications by specifying a template which describes how the application can be materialized. The system automatically discovers and chooses the appropriate services and other resources whose functionality matches the specified application template. The application can be executed after the services and resources have been allocated.

In contrast, interactive application composition enables the users themselves to specify their needs and achieve their goals. For this purpose, the system offers appropriate user tools for composing and controlling application execution at runtime. State of the art user tools are visual editors and graphical user interfaces, as well as solutions relying on physical interfaces and programming-by-example methods.

The automated application composition proposed in this thesis is based on four

prototypes (Publications I-IV). These prototypes range from those that rely on SEA and MGA (Publications I and II) to prototypes with a more sophisticated design which rely on the EA and GA schemas (Publications III and IV). The prototypes optimize the composed applications (the structure and the set of services) which are subject to multiple constraints and user-specific criteria. The prototypes designed also include the graph-based models representing ubiquitous applications, their properties and the environment. The most recent prototype (presented in Publications III and IV) has an extensive model which can be expanded to accommodate additional properties and constraints of applications and the environment. In addition, the optimization criteria can be tailored according to the user's goals without modifying the source code. These automated application composition prototypes were evaluated for performance and other characteristics using synthesized network topologies and also when integrated into the application composition system. The thesis reports on the successful utilization of the developed prototypes also with real applications.

Interactive application composition is studied in this thesis using four prototypes (Publications V-VIII). These prototypes are fully implemented systems with applications. The prototypes enable user tools for interactive application composition which rely on GUIs and physical user interfaces and utilize various contexts. These user tools comprise interaction methods for controlling application composition at runtime, namely manual, context-aware and semi-automated interaction methods (presented in Publications V-VII). In addition, this thesis studies a design approach and UIs for end-user application composition, as presented in Publication VIII. It is argued in this thesis, that a set of methods is preferred to a single one-size-fits-all method, so that users are able to choose the most suitable method according to their needs, the application and various contexts. This finding is supported through evaluation in a series of user experiments which analyzed different aspects of feasibility and usability of the interactive application composition prototypes.

The contributions of this thesis form important steps towards implementing the vision of ubiquitous computing and ubiquitous applications. Particularly application composition studied in this thesis offers concrete solutions which tackle issues related to user acceptance and feasibility of user interfaces and algorithms for application composition.

References

1. McIlroy D (1969) Mass-Produced Software Components. In: Naur P & Randell B (eds) Proceedings of Software Engineering Concepts and Techniques. NATO Science Committee, Scientific Affairs Division, Garmisch, Germany: 138–155.
2. Weiser M (1991) The Computer for the 21st Century. Scientific American 265(3): 94–104.
3. Schuster M, Domene A, Vaidya R, Arbanowski S, Kim M, Lee W & Lim H (2007) Virtual Device Composition. In: Werner B (ed) Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS'07), IEEE Computer Society: 270–278.
4. Mugellini E, Abou Khaled O, Pierroz S, Carrino S & Chabbi Drissi H (2009) Generic Framework for Transforming Everyday Objects into Interactive Surfaces. In: Jacko JA (ed) Proceedings of the 13th International Conference on Human-Computer Interaction, Part III (HCI'09). LNCS 5612: 473–482.
5. Sousa J, 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.
6. Paluska JM, Pham H, Saif U, Chau G, Terman C & Ward S (2008) Structured Decomposition of Adaptive Applications. Pervasive and Mobile Computing 4(6): 791–806.
7. Abowd G, Dey A, Brown P, Davies N, Smith M & Steggles P (1999) Towards a Better Understanding of Context and Context-Awareness. In: Gellersen H (ed) Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99). LNCS 1707: 304–307.
8. Davidyuk O, Riekki J, Rautio VM & Sun J (2004) Context-Aware Middleware for Mobile Multimedia Applications. In: Doermann D & Duraiswami R (eds) Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia (MUM'04): 213–220.
9. Coello Coello CA & Toscano Pulido G (2001) A Micro-Genetic Algorithm for Multiobjective Optimization. In: Coello Coello CA, Hernandez Aguirre A & Zitzler E (eds) Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO'01). LNCS 3410: 126–140.
10. Medina A, Matta I & Byers J (2000) On the Origin of Power Laws in Internet Topologies. ACM SIGCOMM Computer Communication Review 30(2): 18–28.
11. Riekki J, Sánchez I & Pyykkönen M (2010) Remote Control for Pervasive Services. International Journal of Autonomous and Adaptive Communications Systems 3(1): 39–58.
12. Davidyuk O, Sánchez I, Duran JI & Riekki J (2009) CADEAU: Collecting and Delivering Multimedia Information in Ubiquitous Environments, video. In: Adjunct Proceedings of the 7th International Conference on Pervasive Computing (Pervasive'09), Nara, Japan: 283–286.
13. 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.
14. Goddard W & Melville S (2004) Research Methodology: An Introduction. 2nd edition. Juta Academic.
15. Wohlin C, Runeson P & Höst M (2000) Experimentation in Software Engineering: An Introduction. International Series in Software Engineering. Kluwer Academic Publishers.

16. Auer K & Miller R (2001) Extreme Programming Applied: Playing to Win. Addison-Wesley Professional.
17. Pollini A, Grönvall E, Marti P & Rullo A (2006) Constructing Assemblies in the Health Care Domain: Two Case Studies. In: Chittaro L *et al.* (eds) Proceedings of the Italian Workshop of Mobile Guide (as part of Virtuality Conference 2006). Torino, Italy.
18. Rodden T, Crabtree A, Hemmings T, Koleva B, Humble J, Akesson KP & Hansson P (2004) Between the Dazzle of a New Building and Its Eventual Corpse: Assembling the Ubiquitous Home. In: Proceedings of the 5th Conference on Designing Interactive Systems (DIS'04): 71–80.
19. Messer A, Kunjithapatham A, Sheshagiri M, Song H, Kumar P, Nguyen P & Yi KH (2006) InterPlay: a Middleware for Seamless Device Integration and Task Orchestration in a Networked Home. In: Proceedings of the 4th Annual IEEE Conference on Pervasive Computing and Communications (PERCOM'06): 296–307.
20. Newman M, Elliott A & Smith T (2008) Providing an Integrated User Experience of Networked Media, Devices, and Services through End-User Composition. In: Indulska J *et al.* (eds) Proceedings of the 6th International Conference on Pervasive Computing (Pervasive'08), LNCS 5013: 213–227.
21. Masuoka R, Parsia B & Labrou Y (2003) Task Computing - the Semantic Web meets Pervasive Computing. In: Sycara K & Mylopoulos J (eds) Proceedings of the 2nd International Semantic Web Conference (ISWC'03). LNCS 2870: 866–881.
22. Ben Mokhtar S, Georgantas N & Issarny V (2007) COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support. Journal of Systems and Software 80(12): 1941–1955.
23. Sánchez I, Riekki J & Pyykkönen M (2009) Touch&Compose: Physical User Interface for Application Composition in Smart Environments. In: Langer J (ed.) Proceedings of the International Workshop on Near Field Communication: 61–66.
24. Buford J, Kumar R & Perkins G (2006) Composition Trust Bindings in Pervasive Computing Service Composition. In: Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06): 261–266.
25. Lagesse B, Kumar M & Wright M (2010) ReSCo: A Middleware Component for Reliable Service Composition in Pervasive Systems. In: Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'10): 486–491.
26. Lee J, Kim S, Lim H, Schuster M & Domene A (2007) A Software Architecture for Virtual Device Composition and Its Applications. In: Ichikawa H, Cho WD, Satoh I & Youn H (eds) Proceedings of the 4th International Symposium on Ubiquitous Computing Systems (UCS'07). LNCS 4836: 150–157.
27. Ben Mabrouk N, Beauche S, Kuznetsova E, Georgantas N & Issarny V (2009) QoS-Aware Service Composition in Dynamic Service Oriented Environments. In: Bacon J & Cooper BF (eds) Proceedings of the 10th ACM/IFIP/USENIX, 10th International Middleware Conference (Middleware'09). LNCS 5896: 123–142.
28. Sousa JP, Schmerl B, Steenkiste P & Garlan D (2009) Activity-Oriented Computing. Software Applications: Concepts, Methodologies, Tools, and Applications. Hershey, IGI Global, chapter 186: 3215–3241.

29. Kawsar F, Nakajima T & Fujinami K (2008) Deploy Spontaneously: Supporting End-Users in Building and Enhancing a Smart Home. In: Youn HY & Cho WD (eds) Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp'08): 282–291.
30. Vanderhulst G, Luyten K & Coninx K (2008) ReWiRe: Designing Reactive Systems for Pervasive Environments. In: Graham N & Palanque P (eds) Proceedings of the 15th International Workshop on Interactive Systems, Design, Specification, and Verification (DSV-IS'08). LNCS 5136: 155–160.
31. Mavrommati I, Kameas A & Markopoulos P (2004) An Editing Tool that Manages Device Associations in an In-Home Environment. Personal Ubiquitous Computing 8(3-4): 255–263.
32. Mugellini E, Rubegni E, Gerardi S & Khaled OA (2007) Using Personal Objects as Tangible Interfaces for Memory Recollection and Sharing. In: Ullmer B & Schmidt A (eds) Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI'07): 231–238.
33. Chin J, Callaghan V & Clarke G (2006) An End-User Tool for Customising Personal Spaces in Ubiquitous Computing Environments. In: Jin H, Laurence T T & Jeffrey JP (eds) Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC'06). LNCS 4159: 1080–1089.
34. Chin J, Callaghan V & Clarke G (2006) An End-User Programming Paradigm for Pervasive Computing Applications. In: Proceedings of the ACS/IEEE International Conference on Pervasive Services (ICPS'06): 325–328.
35. Mavrommati I & Kameas A (2003) End-User Programming Tools in Ubiquitous Computing Applications. In: Stephanidis C (ed) Proceedings of the 10th International Conference on Human-Computer Interaction (HCI'03): 172–177.
36. Rantapuska O & Lahteenmaki M (2008) Task-based User Experience for Home Networks and Smart Spaces. In: Zaslavsky A & Truong K (eds) Proceedings of the International Workshop on Pervasive Mobile Interaction Devices, Sydney, Australia: 188–191.
37. Newman M & Ackerman M (2008) Pervasive Help @ Home: Connecting People Who Connect Devices. In: Proceedings of the International Workshop on Pervasive Computing at Home (PC@Home), Sydney, Australia: 28–36.
38. Chang SC, Liao CF, Liu YC & Fu LC (2009) A Spontaneous Preference Aware Service Composition Framework for Message-Oriented Pervasive Systems. In: Chang F & Wang Y (eds) Proceedings of the IEEE Joint Conference on Pervasive Computing (JCPC'09): 441–446.
39. Lindenberg J, Pasman W, Kranenborg K, Stegeman J & Neerincx MA (2006) Improving Service Matching and Selection in Ubiquitous Computing Environments: a User Study. Personal Ubiquitous Computing 11(1): 59–68.
40. Rich C, Sidner C, Lesh N, Garland A, Booth S & Chimani M (2006) DiamondHelp: a New Interaction Design for Networked Home Appliances. Personal and Ubiquitous Computing 10(2-3): 187–190.
41. The OASIS Consortium. Web Services Business Process Execution Language (WSBPEL) Version 2.0. URI: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Cited 2012/04/23.
42. The World Wide Web Consortium (W3C). Web Service Conversation Language (WSCL). URI: http://www.w3.org/TR/wscl10/. Cited 2012/04/23.
43. The World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0. URI: http://www.w3.org/TR/wsdl20/. Cited 2012/04/23.

44. Paolucci M, Kawamura T, Payne TR & Sycara KP (2002) Semantic Matching of Web Services Capabilities. In: Horrocks I & Hendler J (eds) Proceedings of the 1st International Semantic Web Conference on The Semantic Web (ISWC'02). LNCS 2342: 333–347.
45. Maedche A & Staab S (2001) Ontology Learning for the Semantic Web. IEEE Transactions on Intelligent Systems 16(2): 72–79.
46. The World Wide Web Consortium (W3C). OWL-S Web Ontology Language: Semantic Markup for Web Services. URI: http://www.w3.org/Submission/OWL-S/. Cited 2012/04/23.
47. Stevenson D, Dutta R, Rouskas GN, Reeves D & Baldine I (2007) On the Suitability of Composable Services for a Next General Assurable Network. In: Proceedings of the 26th IEEE Military Communication Conference 2007, (MILCOM'07): 1–7.
48. Gu T, Pung HK & Zhang DQ (2005) A Service-Oriented Middleware for Building Context-Aware Services. Journal of Network and Computer Applications 28(1): 1–18.
49. Keeney J, Carey K, Lewis D, O'Sullivan D & Wade V (2005) Ontology-Based Semantics for Composable Autonomic Elements. In: Sterritt R, Dobson S & Smirnov M (eds) Proceedings of the Workshop on Artificial Intelligence in Autonomic Communications (as part of the 19th International Joint Conference on Artificial Intelligence, (IJCAI'05)): 1–8.
50. Zhang B, Shi Y & Xiao X (2007) A Policy-Driven Service Composition Method for Adaptation in Pervasive Computing Environment. The Computer Journal 53(2): 152–165.
51. Lupu E & Sloman M (1997) Conflict Analysis for Management Policies. In: Lazar A, Saracco R & Stadler R (eds) Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network Management V: Integrated Management in a Virtual World: 430–443.
52. Wisner P & Kalofonos D (2007) A Framework for End-User Programming of Smart Homes Using Mobile Devices. In: Jiang Y (ed) Proceedings of the 4th IEEE Consumer Communications and Networking Conference (CCNC'07): 716–721.
53. Gross T & Marquardt N (2010) Creating, Editing and Sharing Complex Ubiquitous Computing Environment Configurations with CollaborationBus. Scientific International Journal for Parallel and Distributed Computing 11(3): 289–303. URI: http://www.scpe.org/index.php/scpe/article/view/661/. Cited 2012/04/23.
54. Kandogan E (2001) JigsawTree: Design of a Task Composition Interface for Complex Applications. In: Proceedings of the 8th IFIP International Conference on Human-Computer Interaction (INTERACT'01): 561–568.
55. Cavallaro L, Nitto ED, Furia CA & Pradella M (2010) A Tile-Based Approach for Self-Assembling Service Compositions. In: Calinescu R (ed) Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'10): 43–52.
56. Warth A, Yamamiya T, Ohshima Y & Wallace S (2008) Toward A More Scalable End-User Scripting Language. In: Proceedings of the 6th International Conference on Creating, Connecting and Collaborating Through Computing (C5'08): 172–178.
57. Humble J, Crabtree A, Hemmings T, Akesson KP, Koleva B, Rodden T & Hansson P (2003) Playing With Your Bits: User Composition of Ubiquitous Domestic Environments. In: Dey A, Schmidt A & McCarthy J (eds) Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03). LNCS 2864: 256–263.
58. Want R, Pering T, Sud S & Rosario B (2008) Dynamic Composable Computing. In: Spasojevic M & Corner M (eds) Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile'08): 17–21.

59. Pering T, Want R, Rosario B, Sud S & Lyons K (2009) Enabling Pervasive Collaboration with Platform Composition. In: Tokuda H *et al.* (eds) Proceedings of the 7th International Conference on Pervasive Computing (Pervasive'09). LNCS 5538: 184–201.
60. Boshernitsan M & Downes M (2004) Visual Programming Languages: A Survey. Technical Report UCB/CSD-04-1368. Berkeley CA, University of California. URI: `http://nma.berkeley.edu/ark:/28722/bk0005s5d5t`. Cited 2012/04/23.
61. Mellor SJ & Balcer M (2002) Executable UML: A Foundation for Model-Driven Architectures. Boston MA, Addison-Wesley.
62. Dey A (2005) End-User Programming: Empowering Individuals to Take Control of their Environments. In: Olsen D & Klemmer S (eds) Proceedings of the CHI 2005 Workshop on the Future of User Interface Design Tools. Portland, Oregon, USA.
63. Riekki J (2007) RFID and Smart Spaces. International Journal of Internet Protocol Technology 2(3-4): 143–152.
64. Grönvall E, Marti P, Pollini A & Rullo A (2006) Active Surfaces: a Novel Concept for End-User Composition. In: Mørch A *et al.* (eds) Proceedings of the 4th Nordic Conference on Human-Computer Interaction (NordiCHI'06): 96–104.
65. Zuckerman O, Arida S & Resnick M (2005) Extending Tangible Interfaces for Education: Digital Montessori-Inspired Manipulatives. In: van der Veer G & Gale C (eds) Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05): 859–868.
66. Zuckerman O, Grotzer T & Leahy K (2006) Flow Blocks as a Conceptual Bridge Between Understanding The Structure and Behavior of a Complex Causal System. In: Proceedings of the 7th International Conference on Learning Sciences (ICLS'06): 880–886.
67. Mavrommati I & Darzentas J (2007) End-User Development in AmI: a User Centered Design Overview of Issues and Concepts. e-Minds: International Journal on Human-Computer Interaction 1(3): 87–104.
68. Hardian B, Indulska J & Henricksen K (2008) Exposing Contextual Information for Balancing Software Autonomy and User Control in Context-Aware Systems. In: Zaslavsky A & Truong K (eds) Proceedings of the Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications: 253–260.
69. Balandin S & Waris H (2009) Key Properties in the Development of Smart Spaces. In: Stephanidis C (ed) Proceedings of the 5th International Conference on Universal Access in Human-Computer Interaction (as part of HCI'09). LNCS 5615: 3–12.
70. Michalewicz Z & Fogel D (2004) How to Solve It: Modern Heuristics. 2nd edition. Springer.
71. Kichkaylo T & Karamcheti V (2004) Optimal Resource-Aware Deployment Planning for Component-Based Distributed Applications. In: Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing: 150–159.
72. Kichkaylo T, Ivan A & Karamcheti V (2003) Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. In: Proceedings of the 2003 International Parallel and Distributed Processing Symposium (IPDPS '03): 1–10.
73. Fu X, Shi W, Akkerman A & Karamcheti V (2001) CANS: Composable, Adaptive Network Services Infrastructure. In: Anderson T (ed.) Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'01): 1–12.
74. Wang S, Merrick J & Shin K (2004) Component Allocation with Multiple Resource Constraints for Large Embedded Real-Time Software Design. In: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04): 219–226.

75. Malek S, Mikic-Rakic M & Medvidovic N (2004) A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. In: Dearle A & Eisenbach S (eds) Proceedings of the 3rd International Working Conference on Component Deployment (CD'05). LNCS 3798: 99–114.
76. Ben-Shaul I, Holder O & Lavva B (2001) Dynamic Adaptation and Deployment of Distributed Components in HADAS. IEEE Transactions on Software Engineering 27(9): 769–787.
77. Andrzejak A, Graupner S, Kotov V & Trinks H (2002) Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems. Technical Report HPL-2002-259. Palo Alto, Hewlett-Packard Laboratories.
78. Dionisio DN & Raj R (2006) Partitioning Bin-Packing Algorithms for Distributed Real-Time Systems. International Journal of Embedded Systems 2(3-4): 196–208.
79. Card SK, Newell A & Moran TP (1983) The Psychology of Human-Computer Interaction. Hillsdale NJ, Lawrence Erlbaum Associates.
80. Vastenburg M, Keyson D & Ridder H (2007) Measuring User Experiences of Prototypical Autonomous Products in a Simulated Home Environment. In: Stephanidis C *et al.* (eds) Proceedings of the 12th International Conference on Human-Computer Interaction: Interaction Platforms and Techniques (HCI'07). LNCS 4550: 998–1007.
81. Vanderhulst G, Luyten K & Coninx K (2008) Put the User in Control: Ontology-Driven Meta-level Interaction for Pervasive Environments. In: Bouamrane M, Luz S & Masoodian M (eds) Proceedings of the 1st International Workshop on Ontologies in Interactive Systems (ONTORACT'08): 51–56.
82. Bottaro A, Bourcier J, Escoffier C & Lalanda P (2007) Context-Aware Service Composition in a Home Control Gateway. In: Taïani F & Cerqueira R (eds) Proceedings of the IEEE International Conference on Pervasive Services (ICPS'07): 223–231.
83. Hesselman C, Tokmakoff A, Pawar P & Iacob S (2006) Discovery and Composition of Services for Context-Aware Systems. In: Havinga P, Lijding M, Meratnia N & Wegdam M (eds) Proceedings on the 1st European Conference on Smart Sensing and Context (EuroSSC'06). LNCS 4272: 67–81.
84. Rouvoy R, Barone P, Ding Y, Eliassen F, Hallsteinsen SO, Lorenzo J, Mamelli A & Scholz U (2009) MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. Software Engineering for Self-Adaptive Systems. LNCS 5525: 164–182.
85. Zhou J, Gilman E, Palola J, Riekki J, Ylianttila M & Sun J (2011) Context-Aware Pervasive Service Composition and Its Implementation. Personal Ubiquitous Computing 15: 291–303.
86. Mokhtar SB, Fournier D, Georgantas N & Issarny V (2005) Context-Aware Service Composition in Pervasive Computing Environments. In: Guelfi N & Savidis A (eds) 2nd International Workshop on Rapid Integration of Software Engineering Techniques (RISE'05). LNCS 3943: 129–144.
87. Preuveneers D & Berbers Y (2005) Automated Context-Driven Composition of Pervasive Services to Alleviate Non-Functional Concerns. International Journal of Computing and Information Sciences 3(2): 19–28.
88. Yang Y, Mahon F, Williams M & Pfeifer T (2006) Context-Aware Dynamic Personalised Service Re-composition in a Pervasive Service Environment. In: Jin H, Yang L & Tsai J (eds) Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC'06). LNCS 4159: 724–735.

89. Kalasapur S, Kumar M & Shirazi B (2007) Dynamic Service Composition in Pervasive Computing. IEEE Transactions on Parallel Distributed Systems 18(7): 907–918.
90. Davidyuk O, Sánchez I & Riekki J (2011) CADEAU: Supporting Autonomic and User-Controlled Application Composition in Ubiquitous Environments. Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications, IGI Global, Chapter 4: 74–103.
91. Mukhtar H, Belaïd D & Bernard G (2011) Dynamic User Task Composition Based on User Preferences. ACM Transactions on Autonomous Adaptive Systems 6(1): 4:1–4:17.
92. Ben Lahmar I, Belaid D, Mukhtar H & Chaudhary S (2011) Automatic Task Resolution and Adaptation in Pervasive Environments. In: Bouchachia A (ed.) Proceedings the 2011 International Conference on Adaptive and Intelligent Systems (ICAIS'11). LNCS 6943: 131–144.
93. The World Wide Web Consortium (W3C). Composite Capabilities / Preference Profiles (CC/PP): Structure and Vocabularies. URI: http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/. Cited 2012/04/23.
94. Ben Mokhtar S (2007) Semantic Middleware for Service-Oriented Pervasive Computing. Ph.D. thesis, University of Pierre and Marie Curie (Paris VI), Paris, France.
95. Halevy A (2005) Why Your Data Won't Mix: Semantic Heterogeneity. ACM Queue: Semi Structured Data 3(8): 50–58.
96. Raverdy PG, Issarny V, Chibout R & La Chapelle A (2006) A Multi-Protocol Approach to Service Discovery and Access in Pervasive environments. In: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'06): 1–9.
97. Bottaro A, Gerodolle A & Lalanda P (2007) Pervasive Service Composition in the Home Network. In: Yang L & Ma J (eds) Proceedings of the 21st International Conference on Advanced Information Networking and Applications (AINA'07): 596–603.
98. Dey A & Newberger A (2009) Support for Context-Aware Intelligibility and Control. In: Olsen D & Arthur R (eds) Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI'09): 859–868.
99. David W, William S & Mordechai GO (2010) Facilitating Consumer Acceptance of RFID and Related Ubiquitous Technologies. International Journal of Applied Logistics (IJAL) 1(1): 16–27.
100. Muhammad H & Victor P (2008) Consumer Acceptance of RFID Technology: An Exploratory Study. IEEE Transactions on Engineering Management 55(2): 316–328.
101. Bellotti V & Sellen A (1993) Design for Privacy in Ubiquitous Computing Environments. In: de Michelis G, Simone C & Schmidt K (eds) Proceedings of the 3rd European Conference on Computer-Supported Cooperative Work: 77–92.
102. Langheinrich M (2001) Privacy by Design: Principles of Privacy-Aware Ubiquitous Systems. In: Abowd G, Brumitt B & Shafer S (eds) Proceedings the International Conference on Ubiquitous Computing (UBICOMP'01). LNCS 2201: 273–291.
103. Hong JI & Landay JA (2004) An Architecture for Privacy-Sensitive Ubiquitous Computing. In: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys'04): 177–189.
104. Beresford A & Stajano F (2003) Location Privacy in Pervasive Computing. IEEE Transactions on Pervasive Computing 2(1): 46–55.

105. Campbell R, Al-Muhtadi J, Naldurg P, Sampemane G & Dennis Mickunas M (2003) Towards Security and Privacy for Pervasive Computing. In: Okada M *et al.* (eds) Proceedings of the International Symposium on Software Security - Theories and Systems (ISSS'03). LNCS 2609: 77–82.
106. Meyer S & Rakotonirainy A (2003) A Survey of Research on Context-Aware Homes. In: Johnson C, Montague P & Steketee C (eds) Proceedings of the Australasian Information Security Workshop (as part of the 2003 Conference on ACSW Frontiers - Volume 21): 159–168.
107. Cardoso RS & Issarny V (2007) Architecting Pervasive Computing Systems for Privacy: A Survey. In: Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA'07): 26–30.
108. Riva O (2007) Middleware for Mobile Sensing Applications in Urban Environments. Ph.D. thesis, University of Helsinki. URI: http://urn.fi/URN:ISBN:978-952-10-4288-1. Cited 2012/04/23.
109. Jurmu M (2008) Managing User-Centric, Opportunistic Device Ensembles in Smart Spaces. In: Adjunct Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp'08): 98–102.
110. Gajos K, Weisman L & Shrobe H (2001) Design Principles for Resource Management Systems for Intelligent Spaces. In: Laddaga R, Robertson P & Shrobe H (eds) Proceedings of the 2nd International Workshop on Self-Adaptive Software: Applications (IWSAS'01). LNCS 2614: 198–215.
111. Jurmu M, Kukka H, Hosio S, Riekki J & Tarkoma S (2011) Leasing Service for Networks of Interactive Public Displays in Urban Spaces. In: Riekki J, Ylianttila M & Guo M (eds) Proceedings of the 6th International Conference on Advances in Grid and Pervasive Computing (GPC'11). LNCS 6646: 198–208.
112. Cadenas A, Ruiz C, Larizgoitia I, García-Castro R, Lamsfus C, Vázquez I, González M, Martín D & Poveda M (2009) Context Management in Mobile Environments: A Semantic Approach. In: Proceedings of the 1st Workshop on Context, Information and Ontologies (CIAO'09): 2:1–2:8.
113. Serrano M, Nigay L, Lawson JYL, Ramsay A, Murray-Smith R & Denef S (2008) The OpenInterface Framework: a Tool for Multimodal Interaction. In: Czerwinski M, Lund A & Tan D (eds) Extended Abstracts Proceedings of the 2008 Conference on Human Factors in Computing Systems (CHI'08): 3501–3506.
114. Sousa JP, Schmerl B, Poladian V & Brodsky A (2008) uDesign: End-User Design Applied to Monitoring and Control Applications for Smart Spaces. In: Kruchten P, Garlan D & Woods E (eds) Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA'08): 71–80.

Original articles

I Davidyuk O, Ceberio J & Riekki J (2007) An Algorithm for Task-based Application Composition. In the 11th IASTED International Conference on Software Engineering and Applications (SEA'07), Cambridge, MA, USA, November 2007: 465–472. URI: http://www.actapress.com/Abstract.aspx?paperId=32100.

II Davidyuk O, Selek I, Ceberio J & Riekki J (2007) Application of Micro-Genetic Algorithm for Task Based Computing. In the 1st International Conference on Intelligent Pervasive Computing (IPC'07), Jeju Island, Korea, October 2007: 140–145. DOI: 10.1109/IPC.2007.23.

III Davidyuk O, Selek I, Duran JI & Riekki J (2008) Algorithms for Composing Pervasive Applications. International Journal of Software Engineering and Its Applications 2(2): 71–94. URI: http://www.sersc.org/journals/IJSEIA/vol2_no2_2008/7.pdf.

IV Sánchez I, Davidyuk O & Riekki J (2009) Towards User-Oriented Application Composition. In the IEEE International Workshop on Pervasive Service Computing and Applications (PSCA'09), As part of the 4th International Conference on Frontier of Computer Science and Technology (FCST'09), Shanghai, China, December 2009: 698–704. DOI: 10.1109/FCST.2009.76.

V Davidyuk O, Sánchez I, Duran IJ & Riekki J (2008) Autonomic Composition of Ubiquitous Multimedia Applications in REACHES. In the 7th International ACM Conference on Mobile and Ubiquitous Multimedia (MUM'08), Umeå, Sweden, December 2008: 105–108. DOI: 10.1145/1543137.1543159.

VI Davidyuk O, Sánchez 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, IGI Global, Chapter 4: 74–103. DOI: 10.4018/978-1-60960-611-4.ch004.

VII Davidyuk O, Gilman K, Sánchez I, Mäkipelto J, Pyykkönen M & Riekki J (2011) *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.

VIII Davidyuk O, Georgantas N, Issarny V & Riekki J (2011) MEDUSA: A Middleware for End-User Composition of Ubiquitous Applications. In Mastrogiovanni F & Chong N-Y (eds) Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives, IGI Global, Chapter 11: 197–219. DOI: 10.4018/978-1-61692-857-5.ch011.

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

403. Räsänen, Teemu (2011) Intelligent information services in environmental applications

404. Janhunen, Janne (2011) Programmable MIMO detectors

405. Skoglind-Öhman, Ingegerd (2011) Participatory methods and empowerment for health and safety work : Case studies in Norrbotten, Sweden

406. Kellokumpu, Vili-Petteri (2011) Vision-based human motion description and recognition

407. Rahko, Matti (2011) A qualification tool for component package feasibility in infrastructure products

408. Rajala, Hanna-Kaisa (2011) Enhancing innovative activities and tools for the manufacturing industry: illustrative and participative trials within work system cases

409. Sinisammal, Janne (2011) Työhyvinvoinnin ja työympäristön kokonaisvaltainen kehittäminen – tuloksia osallistuvista tutkimus- ja kehittämisprojekteista sekä asiantuntijahaastatteluista

410. Berg, Markus (2011) Methods for antenna frequency control and user effect compensation in mobile terminals

411. Arvola, Jouko (2011) Reducing industrial use of fossil raw materials : Techno-economic assessment of relevant cases in Northern Finland

412. Okkonen, Jarkko (2011) Groundwater and its response to climate variability and change in cold snow dominated regions in Finland: methods and estimations

413. Anttonen, Antti (2011) Estimation of energy detection thresholds and error probability for amplitude-modulated short-range communication radios

414. Neitola, Marko (2012) Characterizing and minimizing spurious responses in Delta-Sigma modulators

415. Huttunen, Paavo (2012) Spontaneous movements of hands in gradients of weak VHF electromagnetic fields

416. Isoherranen, Ville (2012) Strategy analysis frameworks for strategy orientation and focus

417. Ruuska, Jari (2012) Special measurements and control models for a basic oxygen furnace (BOF)

418. Kropsu-Vehkaperä, Hanna (2012) Enhancing understanding of company-wide product data management in ICT companies

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

UNIVERSITY OF OULU P.O.B. 7500 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-951-42-9837-0 (Paperback)
ISBN 978-951-42-9838-7 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO