Architectural Support for Dynamic Homecare Service Provisioning

Shahin (Alireza) Zarghami

Enschede, The Netherlands, 2013

Ph.D. dissertation commitee:

Chairman and secretary:	Prof.dr.ir. A.J. Mouthaan (University of Twente, the Netherlands)
Promotor:	Prof.dr. R. J. Wieringa (University of Twente, the Netherlands)
Assistant Promotors:	Dr.ir. M. J. van Sinderen (University of Twente, the Netherlands)
Members:	Prof.dr.ir. M. Akşit (University of Twente, the Netherlands)
	Prof.dr. S. Dustdar (Vienna University of Technology, Austria)
	Dr. L. Ferreira Pires (University of Twente, the Netherlands)
	Prof.dr.ir. L.J.M. Nieuwenhuis (University of Twente, the Netherlands)
	Prof.dr. T. Plagemann (University of Oslo, Norway)

CTIT Ph.D. Thesis Series No. 13-257
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

SIKS Dissertation Series No. 2013-26
This research has been carried out under the auspices of SIKS, the Dutch research School for Information and Knowledge Systems.

This work is part of the IOP GenCom U-Care project (http://ucare.ewi.utwente.nl), which is sponsored by the Dutch Ministry of Economic Affairs under contract IGC0816.

Printed and bound by Ipskamp Drukkers B.V. The Netherlands
Cover designed by Hamidreza Darvish
ISSN 1381-3617
ISBN 978-90-365-0077-7
http://dx.doi.org/10.3990/1.9789036500777

ARCHITECTURAL SUPPORT FOR DYNAMIC HOMECARE SERVICE PROVISIONING

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
Prof.dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday the 19th of September 2013 at 14.45

by
Shahin (Alireza) Zarghami

born on 25 March 1978
in Tehran, Iran

Dit proefschrift is goedgekeurd door:
Prof.dr. R. J. Wieringa (promotor) en Dr.ir. M. J. van Sinderen (assistent-promotor)

به شکیبایی و مهربانی پدر و مادر

همدلی و همراهی همسر

شوق و امید فرزند

Abstract

Providing IT-based care support for elderly people at home (i.e., care-receivers) is proposed as a highly promising approach to address the aging population problem. With the emergence of homecare application service providers, a homecare system can be seen as a set of linked services. Configuring and composing existing homecare application services to create new homecare composite applications can reduce the application development cost. The idea even looks more promising if the service provisioning is dynamic, i.e., if applications can update their behaviours with respect to the contextual changes without or with minimum manpower. Dynamic service provisioning can play an important role to accept homecare systems in practical settings.

This thesis proposes a *Dynamic Homecare Service Provisioning (DHSP)* platform to address the homecare contextual changes in an effective and efficient manner. In dynamic service provisioning, a composite application can be reconfigured. This can happen automatically on-the-fly (called adaptive service composition), by end-users for example nurses (which we call tailorable service composition) or by a programmer (which we call evolvable service composition). The proposed DHSP platform provides adaptive, tailorable and evolvable service provisioning in the homecare domain.

To support this, a hybrid service composition approach has been proposed, in which the core of the application logic, which is rather stable, is specified in terms of processes, while rules are employed to specify the conditions and constraints to adapt the application behaviour. The rules are then exposed as a *decision service*, which can be employed by the process to make adaptation decisions with respect to runtime circumstances.

As a proof of concept, we have developed a software prototype of our platform. The prototype was subsequently used in a real-world field test, which consists of two experiments, at a care institution in the Netherlands to validate the approach. The validation included both objective and subjective measurements. Being able to combine objective and subjective measurements, would be useful to know which

level of effectiveness and efficiency is acceptable in the homecare domain. Moreover, we identified explanations of our observations that allowed us to understand which parts of our approach need further improvement.

During the field test, the DHSP platform was used daily with more than 400,000 transactions in total over four months among the infrastructure and application services. The goal of the field test was to study the usability of the DHSP platform to address the homecare contextual changes in terms of (a) $effectiveness$, (b) $efficiency$, and (c) $satisfaction$, both subjectively and objectively.

In the first experiment of our field test, we found out that although the application services as actually delivered by the service providers met the users' requirements, there were architectural mismatches across service providers due to unstated assumptions. Thus we introduced an Assumption-based Risk identification Method (ARM). The ARM method helped us identify several risks before using the DHSP platform in the second experiment of our field test.

During the field test, we observed that the adaptivity of the homecare applications met the end-users' (care-receivers and nurses) expectations, at least in the second experiment. The tailorability of the homecare applications also met the nurses' expectations except for one specific type of homecare application. The nurses were satisfied with the fact that they only needed to use the same tailoring application for all the homecare applications. We also observed that the evolvability of the homecare applications met the programmers' expectations. This was possible mainly because of using the decision service. Our field test showed that using the decision service improves the evolvability while its cost in terms of time and data communication is rather small.

Our conclusion from the field test is that the DHSP platform is suitable for homecare service provisioning. However, we only evaluated the proposed DHSP platform with a limited number of participants (care-receivers, care-givers, IT specialist) in one care center in the Netherlands in which care-receivers live in their care homes. Evaluation of the platform in other situations (e.g., a situation where care-receivers live at their own homes and receive support remotely) may have different results. Moreover, the platform should be evaluated using other homecare applications and their required application services.

Acknowledgements

"We are waves whose stillness is non-being"
— Kalim Kashani

Writing this acknowledgement was difficult for me even more difficult than the other chapters of this dissertation. It was truly a challenge for me to express feeling and gratitude for my lovely colleagues and friends by only a few words.

I have done my thesis in Information System (IS) group at the University of Twente where I had the chance to work with many wonderful people.

Suse, I believe you are absolutely one of a kind. You have been always supportive and caring to me indeed like a very kind mother. Seeing you every morning always reassured me that everything would be ok today. I never forgot your time and effort to contact different energy companies and agencies just to make sure that our rights are being respected. I truly appreciate all your help and support.

Roel, you taught me how a simple pencil and paper could be powerful. You helped me to understand that we do not need any fancy technologies to think perfectly and that all we need to think is already embedded within our mind. You taught me to practice being clear and concrete in proposing my ideas, which was quite helpful for my PhD research and even for my personal life.

Marten, you taught me how to have a smooth discussion over a complicated issue, how to look at the issue from different points of view, and how to deal with that by taking only small steps. I believe that I have been so lucky to have a rational but also a very kind and emotional supervisor like you.

Mohammad, you have been truly my caring colleague, wonderful friend, and kind brother. I believe I could not finish my PhD thesis without your help and support particularly as I had to finish my PhD

sooner than the standard four years duration. I am so happy that I have the opportunity to work with you again in Accenture. Having you and your lovely family, Leila and Araz here close to us always have been reassuring me indeed.

I would like to thank to all my present and past colleagues from the IS group, who have been more like a friend to me: Silja, Hassan, Steven, Andre, Andreas, Camlon, Chen, Eelco, Klaas, Lianne, Luis, Maya, Nelly, Pascal, Sergio, Wilco, Wolter, and Zornitza. I am also deeply thankful to Eelco, and the effort he made in writing and discussing a problem patiently. I appreciate all the support from Brama particularly in the beginning of my PhD for helping me how to write a scientific paper.

I am more than grateful to the members of my defense committee and in particular Luís Ferreira. I learned a lot from your comments and I believe my dissertation has quite improved by applying your comments.

This work is part of the IOP GenCom U-Care project (http://ucare.ewi.utwente.nl), sponsored by the Dutch Ministry of Economic Affairs. I would like to extend my gratitude to all its partners. I would like to thank Jan-Willem, Lucas, Maria and Bert-Jan, Mobihealth particularly Tom and Daniel; and Orbis specially Cindy, John, Lilian and Maarten.

I would also like to thank all my friends in the Netherlands: Afshin and Elham, Alireza and Zahra, Meysam and Hajar, Hamed and Maryam, Gaby and Freek, Robbie, Majid, Mohammadreza, Ida, Jan-Pieter, and Victor. Catherine and Kees, you taught me how to enjoy each and every moment of life. To me, you are truly symbol of peace and happiness. Irma and Arjen you are indeed a wonderful family. I learnt from you how to live life in harmony and peace, and how we can simply cheer up by any coming events and news in life.

The last but not least, I would like to thank Soude, my caring and supportive beloved wife and Ava, my lovely daughter who cheered me up whenever I felt down during my PhD. I truly would like to express my gratitude to Shokooh and Hadi, my lovely parents, for their endless support and patience in my whole life. I am also more than grateful to Homa for her unique positive attitude that always lightened me, and to Alireza for all his kindness. I appreciate my wonderful brother, Shahrokh, for his wise advices and support. I also would like to thank my lovely sisters, Shirin and Maryam, particularly for being very caring aunties for Ava.

Shahin Zarghami
Delft, August 2013

Contents

Chapter 1

Introduction

"An education isn't how much you have committed to memory, or even how much you know. It's being able to differentiate between what you do know and what you don't."
— Anatole France

This thesis proposes an architectural support for dynamic service provisioning in the homecare domain by introducing a *Dynamic Homecare Service Provisioning (DHSP)* platform. People needing homecare applications, such as patients or some elderly people, are generally called care-receivers. Care-receivers are subject to different types of contextual changes, such as changes in their vital-signs (e.g., blood pressure value), location (e.g., inside or outside home) and their capabilities (e.g., sight, hearing). Thus many (un)foreseen changes may occur during the provisioning of a homecare application and the application behaviour should be adapted accordingly.

A DHSP platform should address the contextual changes without or with minimum manpower. A homecare application running on the DHSP platform should adapt its behaviour with respect to the contextual changes. This can be done automatically by the homecare application itself (which we refer to as adaptivity), by an end-user, for example a nurse (which we refer to as tailorability), or by a programmer who designed the application (which we refer to as evolvability). The aim of our research is to provide a DHSP platform for provisioning of composite service-oriented applications in the homecare domain. As such, the platform integrates different IT-based services for the provisioning of homecare applications while supporting their *adaptivity*, *tailorability*, and *evolvability*.

This chapter is organized as follows: Section 1.1 explains the background and the current situation in the homecare domain. Section 1.2 presents an application scenario to motivate the work

presented in this thesis and to clarify our discussion. Section 1.3 explains our motivations and the main challenges. Section 1.4 outlines the main research objective. Section 1.5 presents the research questions. Section 1.6 elaborates on the scope of this thesis. Section 1.7 explains the research methodology adopted in this thesis, and finally Section 1.8 presents the structure of the thesis.

1.1 Background

The population of many Western-European countries have an increasing percentage of elderly people and, consequently, the healthcare-related cost for these countries is growing [53, 97]. According to the European Union Health portal, "*By 2050, the number of people in the EU aged 65 and above is expected to grow by 70% and the number of people aged over 80 by 170%. This raises important challenges for the 21st century ... adapt health systems to the needs of an aging population*" [1]. In the Netherlands, in 2003, 14% of the Dutch population was over 65 years of age and this number is expected to grow up to 22% in 2030 [135]. Currently care services for elderly persons, i.e., care-receivers, are provided (mainly) manually and by qualified healthcare staffs, i.e., care-givers. Due to the aging population and lack of professional care-givers, in the near future, IT-based system can play an important role in providing care services [58].

Providing automated care support for elderly in their own home, i.e., homecare system, is proposed as a highly promising solution to alleviate the problem of the growing healthcare related cost for elderly [58]. A homecare system is "*a potentially linked set of services ... that provide or support the provision of care in the home*" [101]. The benefits of homecare systems are not limited to financial aspects but also from social perspective: (a) to have an independent life as long as possible, and (b) to stay in their own home instead of living in a care center [105]. Moreover, the homecare systems would improve the quality of care by monitoring the elderly continuously (24 hours / 7 days) in their own homes [91].

The recent progress in ubiquitous computing [153], remote patient monitoring and treatment [22], and domotic applications [6] leads to many useful end-user applications and technologies for the homecare services provisioning, such as vital signs monitoring and event-based alarm [103, 59]. Although several promising prototype developments have been done in academia [98, 110, 21, 55], yet, there are many challenges and hurdles to overcome before having realistic homecare solutions [137, 169, 10].

1.2 Our Application Scenario

We use the following homecare application scenario to motivate the work presented in this thesis and to clarify our discussion.

1.2.1 Homecare Situation and Needs of Jan and Marie

Jan and Marie are elderly people who live in their apartments located in a care center. The aim of the care center is to provide round-the-clock services to elderly people and at the same time to enable them to live an independent life as much and as long as possible. As such, their vital-signs are measured regularly. For example, their weight and blood pressure values are measured once and twice a day, respectively. When these vital-sign values are not in a normal range, then an alert is sent to their nurses. They also need to take medication regularly. Additionally, Jan has a hearing disorder while Marie has a vision impairment. Besides that, both of them suffer from amnesia and need to be reminded of their tasks (e.g., taking medication or measuring blood pressure).

1.2.2 Introducing IT-Based Homecare Support Solution

Nancy is a professional nurse responsible to (re)tailor the homecare applications installed in their apartments through a tailoring application. A programmer provides the tailoring application to enable Nancy to (re)tailor homecare applications on her Tablet PC. If Nancy cannot support new requirements through the tailoring platform, the programmer modifies the corresponding logic of the homecare applications and the tailoring application respectively to support new tailoring possibilities. The apartments are equipped with a Tablet PC, a PDA, a medicine dispenser and vital-sign monitoring devices (i.e., a weight scale, blood pressure measurement devices and oxygen saturation meter). Nancy has a PDA and Tablet PC to tailor and to interact with the homecare applications.

1.2.3 Nancy Tailors Homecare Applications

Nancy tailors two homecare applications for both Jan and Marie, (1) to remind them to measure their vital-signs, i.e., vital-sign monitoring (VsM), and (2) to remind them to take their medicines on time, i.e., medication monitoring (MdM). These applications send an alert to care-givers if Jan and Marie forget to measure their vital-signs or to take their medicine. They also send an alert if the measured vital-sign values are either higher or lower than the specified threshold values.

Nancy tailors the two homecare applications to send reminder messages earlier if Jan and Marie are outside their apartments.

Moreover, these two applications are tailored to use textual messages on Tablet PC for Jan and audio with high voice volume on PDA for Marie as the reminder.

As soon as Nancy finishes tailoring an homecare application, the tailoring application configures the corresponding applications and devices. For instance, the tailoring application configures a calendar application running on Jan's Tablet PC to show him the scheduled medication time, and the medicine dispenser device to enable dispensing the medication at the scheduled time.

1.2.4 Jan and Marie Use the Homecare Applications

After Nancy has created the homecare applications tailored to the needs of Jan and Marie, the applications are deployed, activated, and provisioned to them. We explain two scenarios as follows:

– *Scenario 1:* at some point in time, Jan is outside his apartment, thus the application sends a reminder message earlier than the scheduled time. However, Jan cannot reach his apartment to measure his blood pressure at that scheduled time and an alert is sent to Nancy's PDA. Therefore, Nancy re-tailors the application by increasing the time in advance to send a reminder for Jan and the application behaviour will be updated immediately. Next time, Jan measures his blood pressure after the second reminder and then the application stops sending reminders and shows the measured value on Jan's and Nancy's Tablet PCs.

– *Scenario 2:* at some point in time, Marie measures her blood pressure and the result is higher than the threshold, which is set by Nancy, and then an alert is sent to Nancy's PDA. After receiving the alert, she goes to the Marie's apartment. However, the blood pressure value is not much higher than normal and it is not necessary to do anything. Nancy discusses the problem with the care center management board and they decide to have the measured vital-sign values with the alert messages. Since, this new functionality can not be defined by the existing tailoring application, the care center asks the programmer to add a new tailoring feature to the VsM application. After the programmer updates the application, Nancy re-tailors the applications for both Marie and Jan to enable the new feature. Later, Jan and Marie's running homecare applications for Jan and Marie send their blood pressure values in the alert messages.

1.3 Motivation and Challenges

In todays' competitive business environment, it would be highly desirable to reduce the development time and cost of application development for addressing new business requirements. With the emergence of the Service-Oriented Architecture (SOA) paradigm, composing applications by reusing existing services is expected to lowers development cost [83]. The used services can be provided by different organizations, and programmers can use them to build a composite application without detailed knowledge of their internal implementation mechanisms.

Similarly, in the homecare domain, many useful components and technologies have recently emerged [103, 59]. Therefore, an integration solution that can bring all these components and technologies together seems more feasible than any time before. Service-Oriented Architecture with its service composition principle provides a promising approach for homecare service provisioning that support seamless cooperation of heterogeneous technology solutions. Several efforts to define such a service-oriented platform have been undertaken [98, 110, 170, 21].

By a service provisioning platform, we mean that the platform should be able to integrate different IT-based services for the provisioning of homecare applications. By dynamic service provisioning platform, we mean that the platform should also support the three forms of dynamicity: adaptivity, tailorability, and evolvability of the homecare applications running on this platform. Our motivations to design a *dynamic homecare service provisioning (DHSP) platform* are explained as follows:

1. **Interoperability**: a homecare composite application may employ several application services from different service providers [104]. These application services most probably have different operations that use different syntax, semantics and exchange patterns for their messages [24, 119, 111, 30]. Nevertheless, the DHSP platform should provide interoperability among these services, i.e., enable them "*to exchange information and to use the information that has been exchanged*" [76].

2. **Service layer integration**: the DHSP platform should provide technology-independence, in the sense that it should shield the application programmer from underlying software and network technologies employed by service providers. The platform abstracts the application services into several basic functions that

can be used by application developers at design time to build composite applications [130, 152, 40]. These basic functions are mapped to concrete application services at execution time and end-users use these application services as a seamless composite application without noticing that they are provided by different service providers.

3. **Dynamicity**: the DHSP platform should support adaptation to the contextual changes in the homecare environment [90, 101]. The contextual changes coming from the care-receiver, homecare applications and environment, can cause dynamicity[100]. To address these contextual changes, the running homecare applications should adapt their behaviours with respect to the contextual changes. This can be done by the homecare application itself, by an end-user for example a nurse, or by a programmer.

We see the following challenges in the homecare domain, which should be addressed by a dynamic homecare service provisioning platform:

1. **Distributed service providers** [101]: a homecare composite application may consist of several application services which are provided by various service providers located in different places (even different countries). Providing location transparency and hiding the complexity of underlying communication infrastructure are challenging. [26]

2. **Heterogeneous service providers**: Interoperability is hard to achieve if composite applications are made up of heterogeneous services that are developed, managed and owned by different organizations [104]. From the software point of view, application services might follow different protocols for discovery, interaction and communication. From the hardware point of view, the physical devices can have different capabilities, varying from a resource-constrained sensor to a computing device with powerful computation resources.

3. **Safety-critical environment**: the homecare domain is a safety-critical environment [94]. It means that incorrect behaviour or the malfunction of an application could lead to loss of life. The application behaviour must comply to what has been defined as its application logic. Behavioural adaptation must be done accurately and within a certain time. Wrong or late adaptation of application

behaviour might be life-threatening.

4. **Medical protocols**: the care-givers are also obliged to follow the existing medical guidelines and protocols employed by the care centers who provide care for elderly [56, 64]. These protocols and guidelines improve the quality and accuracy of care services. Thus a DHSP platform and the homecare application running on top of it must comply with the medical protocols.

5. **Limited resources**: the care centers usually have limited human and system resources to provide the care services [10]. For instance, for the limited human resources, the (re)deployment of a tailored homecare application should take as less time as possible. For the limited system resources, assuming a dedicated provisioning server for each elderly apartment to host the provisioning platform is not realistic, since there is usually one or two care-receivers for each care home. Moreover, the DHSP platform should support on-demand elastic service provisioning, to adjust the IT infrastructure costs to the number of care-receivers.

6. **Privacy**: the homecare system must protect the personal information of care-receivers (e.g., vital-sign values) and control who can access which personal and health-related information [61, 116].

1.4 Objective

Our objective is to design a DHSP platform to address homecare contextual changes in an effective and efficient manner. There are several existing works dealing with supporting IT-based homecare systems [98, 110, 21, 170, 15]. However, little work has been done on dynamic service provisioning and on the validation of such systems in a real-world setting.

We aim to provide a DHSP platform that can provide three usability goals. These three goals are related to three properties of our DHSP platform, namely *adaptivity*, *tailorability* and *evolvability*:

– To adapt a homecare application successfully within a certain time expected by an end-user.

– To deploy a (re)tailored homecare application successfully within a certain time expected by a care-giver.

– To deploy an evolved homecare application successfully within a certain time expected by a programmer.

To evaluate the usability of our DHSP platform, we follow the ISO 9241-11 definition of usability, which is the "*extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*" [80].

In this thesis, we determine how behaviours of the homecare applications can be updated without or with minimum manpower. Addressing foreseen contextual changes (e.g., care-receivers' vital-signs, location, and their capabilities) in the homecare domain has been considered a challenge [90, 101]. In addition, care-receivers use homecare applications in their own ways in an uncontrolled environment [29]. Therefore, several foreseen and unforeseen contextual changes might happen during a homecare application provisioning. As such, we aim at the following properties of our DHSP platform:

1. **Adaptivity**: The homecare applications should be able to monitor contextual changes at runtime and adapt their behaviours accordingly based on their application logic. The application logic is defined by application programmers using context-aware services. Therefore, an application should adapt its behaviour by selecting different application services for a specific Service Building Block (SBB) or configuring and composing the selected services differently based on runtime contextual values. To do so, the DHSP platform should support adaptivity of running applications.

2. **Tailorability**: The homecare application may not be able to monitor some contextual changes such as a decrease in the hearing and sight capabilities of a care-receiver. However, a care-giver can observe these changes. Thus, the suitable responses to these changes (e.g., different audio volume for reminder) can be implemented and then selected or even can be automatically generated based on high-level instructions by an end-user. The DHSP platform should adapt the behaviour of running applications whenever a care-giver (re)tailors these applications (using a tailoring platform) without interrupting other running applications and with minimum time overhead.

3. **Evolvability**: The homecare domain is a multidisciplinary and highly dynamic environment. Thus many (un)foreseen changes may occur during the provisioning of a homecare application. Therefore, at a later phase in the lifecycle of an application, it might be necessary that a programmer of that application modifies the application logic to deal with unforeseen changes. The DHSP platform should support programmers to modify applications cost-effectively and without interrupting the running applications.

1.5 Research Questions

In this thesis, we aim to address three top-level research questions. Each of these questions are decomposed into several sub-questions. Table *1-1* shows the three top-level research questions and their corresponding sub-questions. These questions are categorized as knowledge questions (labeled with "K") and design problems (labeled with "D") [156]. Knowledge questions emerge when we do not know something about a phenomena in the world while design problems arise when we want to change something in the world to (better) satisfy some goals of stakeholders. These knowledge questions are either conceptual questions (labeled with "C"), which will be answered by defining a conceptual model, or empirical questions (labeled with "E"), which will be answered via literature study and interview with practitioners.

In RQ 1, we improve our knowledge about homecare service provisioning, the existing provisioning platforms and their benefits and drawbacks by answering some knowledge questions. Based on the answer of RQ 1.2, we find out that addressing the dynamicity in the homecare domain is a highly-demand requirement, however it has not attracted enough attention. Therefore, we focus on the dynamicity and its related challenge in the homecare domain. As such, we define dynamicity and develop a DHSP platform to address the dynamicity challenge.

In RQ 2, we design a dynamic home service provisioning platform to improve the current homecare service provisioning situation. In the first sub-question, we answer an empirical question to see which of the existing dynamic service provisioning approaches can be used to develop our platform. In the second sub-question, we design the components of the platform and a library of SBBs for the common homecare application services. By designing these SBBs, we aim to shield the application developer from underlying software and network technologies while facilitating the integration of application services.

Table 1-1 The research questions of this thesis (K=knowledge, D=design, C=conceptual, E=empirical

RQ 1 (K): Why do we need a DHSP platform?
- RQ 1.1 (K,C): What is a homecare service provisioning platform?
- RQ 1.2 (K,E): What are the requirements on this platform?
- RQ 1.3 (K,C): What is dynamicity?
- RQ 1.4 (K,C): What is a homecare system?

RQ 2 (D): How to design a DHSP platform?
- RQ 2.1 (K,E): What are the existing dynamic provisioning approaches?
- RQ 2.2 (D,C): What are the components of a DHSP platform?
- RQ 2.3 (D,C): How do these components interact with each other?
- RQ 2.4 (D,C): How to identify the risks of using a DHSP platform?

RQ 3 (K): What is the contribution of our DHSP platform?
- RQ 3.1. (K,E): How to validate the adaptivity of the applications?
- RQ 3.2. (K,E): How to validate the tailorability of the applications?
- RQ 3.3. (K,E): How to validate the evolvability of the applications?

In the third sub-question, we dive into the platform and explain how the (infrastructure) components interact with each other for homecare application provisioning. Like any new design, using our DHSP platform could introduce a new set of risks. Therefore, in the fourth sub-question,we develop a method to identify these new risks. This is very crucial since our platform will be used in the homecare domain as a safety-critical environment.

In RQ 3, we investigate the contribution of our proposed DHSP platform by conducting a field test and observing how the homecare service provisioning situation is affected. Our contribution can be investigated by three empirical sub-questions. In the first sub-question, we investigate if the composite homecare applications running on our platform adapt their behaviour as desired (by care-givers and care-receivers) within a specific time without systems errors. In the second sub-question, we investigate if the DHSP platform adapts the behaviour of running applications as tailored by the care-givers within a specific time without systems errors, i.e., if care-givers, using a tailoring platform, can tailor applications time-effectively. In the third sub-question, we investigate if our approach facilitates the evolvability of homecare composite applications by reducing the time needed to deploy an evolved application by a programmer. These questions are answered through interviews with care-receivers and care-givers (subjectively)

and also system logs (objectively).

1.6 Scope

Our main objective is providing a DHSP platform to support adaptive, tailorable and evolvable homecare composite applications. We mainly focus on designing an application logic structure (we called it service plan) and an architectural support to execute it. A service plan consists of one or more service building blocks (SBBs) and describes the configuration and orchestration of instances of these service building blocks with respect to run-time circumstances. The design and execution of the service plan are related to several research topics, which are not in the scope of this thesis:

– *Application tailoring*: a tailoring platform enables care-givers to create or to tailor service plans that satisfy individual requirements of care-receivers. The service plan supports a set of possible variations. A care-giver as an domain expert, can select of one of these variations by setting the values of the configuration parameters through a tailoring platform. If the implemented variations are not sufficient, an application programmer must manually create new variations. In this thesis, we exclude how new variations can be automatically generated by the DHSP platform upon the request of a care-giver [13, 5].

– *Correctness of the service plan*: a service plan consists of configuration parameters and orchestrations of application services. The DHSP platform checks if all the values of the required configuration parameters of a service plan are given by the tailoring platform in order to execute that service plan. However, the correctness of a service plan is the responsibility of the tailoring platform provider [145].

– *Semantic reasoning*: Using some of the existing dynamic service provisioning approaches, the composite applications can automatically adapt to unforeseen changes by semantic reasoning by using ontologies [138, 54]. Our research does not consider such adaption, and assumes that if an unforeseen change occurs, a application programmer or a care-giver modifies the running application manually. Instead, we focus on facilitating these manual modifications. This inspired by safety-criticalness of the homecare environment that gives a bias to correctness instead of completeness

of application adaptations.

– *Contextual modeling*: The provisioning platform should maintain a data model of the contextual environment in order to support the adaptivity of the composite applications. We do not consider how the environment, including context of end-users, is modeled [39, 68]. Instead, we focus on how this data model can be updated and accessed by different services so that the composite applications can adapt their behaviours within a certain time.

1.7 Research Methodology

We address the research questions of this thesis according to the Technical Action Research (TAR) methodology [158]. We validate the DHSP platform in a field test. For this field test, we follow the checklist introduced by Wieringa [157]. In the TAR methodology, an artifact interacts with a problem context to produce effects, and these effects are evaluated with respect to design criteria. As shown in Fig. *1-1* the artifact in our research is the DHSP platform which we design, implement and validate to interact with the problem context of the homecare domain.

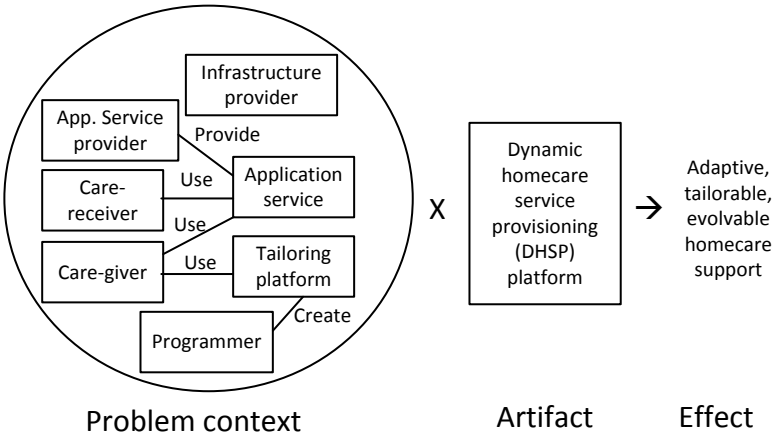

Figure 1-1 : The artifact, context and effect in our research

The problem context consists of several stakeholders such as care-receivers and care-givers, and the environment in which the stakeholders are using the platform and its applications. Care-receivers live in apartments (located in a care center) equipped with

IT-based homecare application services. These application services, for instance a blood pressure measurement service, are provided by third-party service providers. Care-givers use the tailoring platform, which is provided by a programmer, to create a homecare application using the installed application services. The tailored homecare applications are deployed on the DHSP platform to be executed. The infrastructure consists of hardware and software components provided by an infrastructure provider to be used by the DHSP platform and the application services.

According to our motivations, we evaluate the effects of our provisioning platform on the adaptivity, tailorability and evolvability of homecare applications. These effects are the software qualities which we are interested in, and we evaluate them based on the following criteria:

(a) *Effectiveness*: The number of useful system tasks, such as sending an alert (adaptivity goal), deploying a service plan (tailorability goal) or modifying the application logic (evolvability goal), which have been completed within a specific time without systems errors (investigated through system logs).

(b) *Efficiency*: How long it takes for the DHSP platform to complete the system tasks (investigated through system logs).

(c) *End-user satisfaction*: The perceived effectiveness and perceived efficiency of the DHSP platform from the end-users' point of view (investigated through interviews with care-givers and care-receivers).

Satisfaction has some other aspects which are more related to the interaction with end-users [57]. However, our DHSP platform interacts indirectly with end-users through application services or the tailoring platform. Thus, to evaluate the satisfaction of the DHSP platform, we only consider the perceived effectiveness and efficiency. Moreover, we are interested to determine how satisfaction can be affected by integrating different application services.

We follow the engineering cycle introduced in the TAR methodology to carry out our research in this thesis. As shown in Fig. *1-2*, we address the identified research questions in five steps:

1. **Problem investigation**: For problem investigation we perform the following actions:

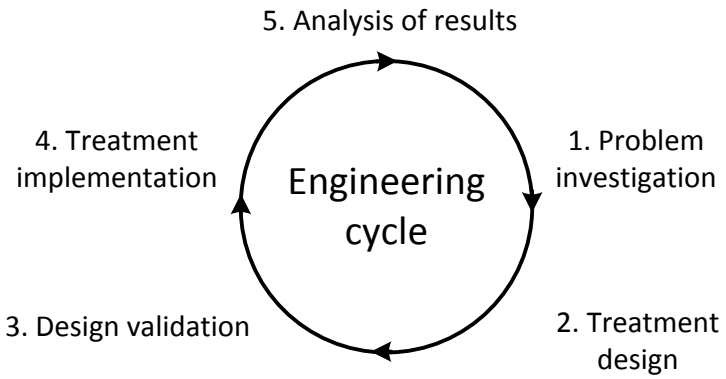

Figure 1-2 : The engineering cycle used in this thesis, adopted from [158]

- *Identification of the requirements on the DHSP platform in the homecare domain*: We interview care-givers to identify a list of requirements on the DHSP platform. Moreover, we perform a literature study on existing homecare systems and the requirements these systems were expected to address. Then we discuss with an application programmer, who provides a tailoring platform for common homecare applications, to understand how applications should be created and tailored. These applications are selected by a programmer based on his interview with the care-givers. We also discuss with the service providers to know their requirements in order to integrate their application services with the DHSP platform. Finally, we list the requirements which used in this thesis to be fulfilled by the DHSP platform and evaluated in the field test.

- *Identification of existing service provisioning approaches*: We perform a literature study on existing service provisioning approaches, both homecare specific and domain-independent approaches, to determine which one of them is more suitable in the homecare domain with respect to the identified requirements.

- *Study existing risk identification methods applicable for our DHSP platform*: We perform a literature study on existing risk identification methods which can identify the risks of using the DHSP platform with respect to different service providers and their individual requirements.

2. **Treatment design**: based on the result of the problem

investigation, we design the following components:

- *Library of SBBs*: With respect to our selected homecare applications and service providers, we make a list of service building blocks (SBBs) and their configuration parameters to shield the application programmer from underlying software and network technologies.

- *Orchestrations patterns*: For each homecare application, we provides several orchestrations patterns to compose the application services based on the runtime circumstances. These orchestrations can be selected by the provisioning platform at runtime with respect to the values of the configuration parameters and runtime circumstances.

- *Assumption-based risk identification method*: Based on the result of *Identification of existing risk identification methods applicable for our DHSP platform* study, we introduce an Assumption-based Risk identification Method (ARM). The method identifies potential risks of using the DHSP platform due to mismatched unstated assumptions made by different service providers, and this mismatch can potentially causes an incorrect composite application to be delivered to end-users. The method helps us identify several risks before using the DHSP platform in the field test.

- *Service provisioning approach and its supporting architecture*: Based on the result of *Identification of existing service provisioning approaches*, we design a service provisioning approach (we call it *Decision as a service*) and the DHSP platform as its architectural support.

3. **Design validation**: Before prototyping the DHSP platform, we go back to application programmers and service providers, and interview them as follow:

   - *Interview on the design of SBBs and orchestration patterns*: We perform several in-depth interviews [89] with both programmers and application providers to investigate if the DHSP can interact with the provided services. This study helps us select a suitable interaction pattern (synchronous or asynchronous) for each service interaction, and complete the configuration parameters and their possible variations.

- *Interview using ARM*: We perform risk assessment using ARM by performing several in-depth interviews [89] with service providers and application programmers to identify and to analyze potential risks of using our approach and consequently to identify further requirements on the DHSP platform.

4. **Treatment implementation**: To show the feasibility of the proposed approach, we develop a prototype of the DHSP platform to be tested in an elderly care institute as our field test. To validate the approach, we perform a field test in two series of experimental studies with a total duration of 4 months.

5. **Analysis of results**: The experiments are conducted in a near real-life setting at the care institution and after each series of the experiments, we perform interviews with the care-givers. The interviews are analyzed to evaluate the usability of the DHSP platform in terms of its perceived *effectiveness* and *efficiency* (we call it *satisfaction*) and to improve the platform. We also analyze the system logs with more than 400000 transactions among the application services to objectively investigate the *effectiveness* and *efficiency* of the DHSP platform.

1.8 Structure

In this thesis, each chapter starts with a short introduction and the reference to the paper(s) which the chapter is based on. Figure *1-3* presents the structure of this thesis, indicating how the chapters cover the above mentioned research methodology and answer the research questions. We introduce the chapters of this thesis as follows:

- *Chapter 1 - Introduction:* provides an introduction of this thesis by presenting background information, motivation, objectives, research questions, research methodology, scope and a short introduction of the approach which we have used.

- *Chapter 2 - Dynamic Homecare Service Provisioning:* defines the basic terminology and fundamental concepts which we have used in this thesis. Moreover, it introduces a framework to explain different challenges and functionalities of a homecare service provisioning platform. The framework can be used to compare

existing homecare service provisioning platforms. This chapter is partly based on papers [167, 9].

– *Chapter 3 - Related work:* positions our concepts with respect to related existing terminology and provides an overall view of existing dynamic service provisioning approaches. Besides, it explains some of the existing homecare systems using the framework introduced in Chapter 2 and motivates our decision to focus on the dynamicity challenges. This chapter is based on the related works of the papers mentioned in other chapters.

– *Chapter 4 - Requirements:* provides a list of requirements which we have identified through interviews and literature study on the DHSP platform, and discusses which of the existing dynamic service provisioning approaches suit the identified requirement the most. This chapter is partly based on paper [10].

– *Chapter 5 - Dynamic Homecare Service Provisioning Platform:* explains the components of the DHSP platform (e.g., infrastructure services), a library of SBBs and their configuration parameters, and how a tailoring platform deploys a service plan to the DHSP platform. It reflects the result of our interviews to validate the design of SBBs, and consequently how we make several design choices with respect to the requirements identified in Chapter 4. This chapter is partly based on paper [8].

– *Chapter 6 - Decision as a Service:* introduces our dynamic service provisioning approach as the core of the DHSP platform and explains how the infrastructure services such as orchestration and decision services interact with each other. This chapter is partly based on paper [165].

– *Chapter 7 - Assumption-based Risk Identification Method:* introduces an Assumption-based Risk identification Method (ARM). The method identifies potential risks of using the DHSP platform due to mismatched unstated assumptions made by different service providers. Moreover, it positions the ARM method with respect to existing risk identification methods and explains how the method is applied between the two experiments through a set of interviews, and consequently the risks which have been identified. This chapter is partly based on paper [166].

- *Chapter 8 - Experimental Prototype:* presents the prototype implementation of the DHSP platform.

- *Chapter 9 - Validation:* presents two experimental studies using the prototype in our field test. Using the results obtained from the field test, we evaluate and validate the prototype of the DHSP platform. This chapter is partly based on paper [11].

- *Chapter 10 - Conclusions and Future Work:* reflects on the work presented in this thesis. It discusses lessons learned and reusable results in other application domains. It further presents some additional challenges in the area of dynamic service provisioning in the homecare domain and outlines potential future research directions.

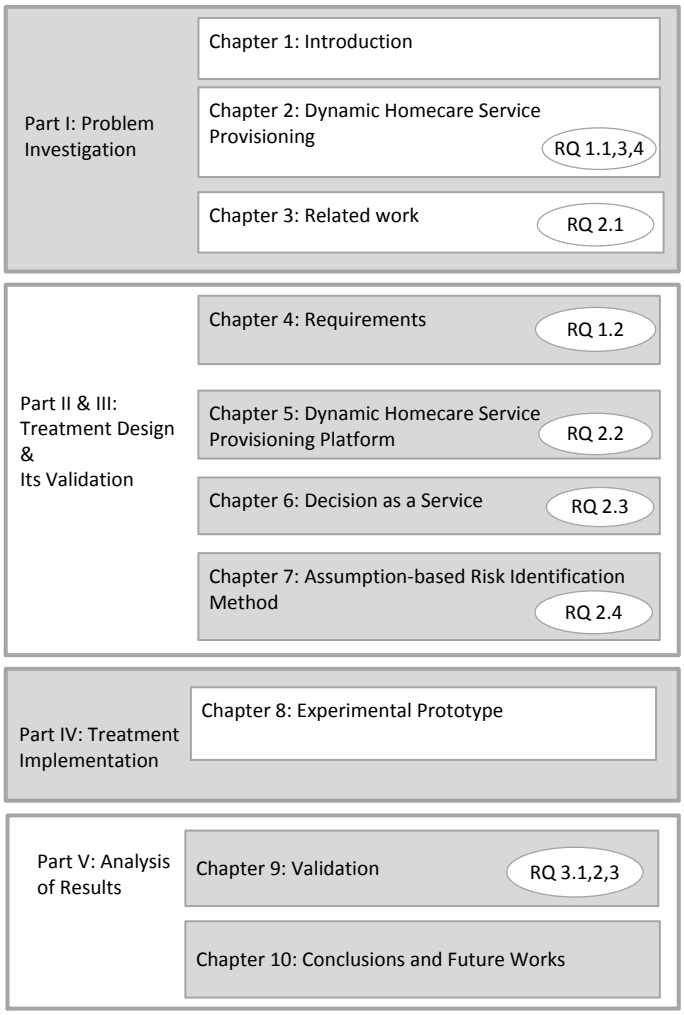

Figure 1-3 : Thesis structure: the chapters, research questions and research methodology.

Chapter 2

Dynamic Homecare Service Provisioning

"If I had an hour to save the world. I would spend 59 minutes defining the problem and one minute finding solutions."
— Albert Einstein

This chapter defines the basic terminology and fundamental concepts which we have used in this thesis. For example, it defines the distinction between different types of contextual changes and how they should be addressed. We further introduce the concept of dynamic provisioning platform and explain how a tailoring platform communicates with our provisioning platform to create or to tailor a composite application. As such, we need to introduce the stakeholders and to explain how they interact with these two platforms. Then we specialize the introduced dynamic provisioning platform as a dynamic homecare service provisioning (DHSP) platform including its stakeholders for the homecare domain. Moreover, we consider the SOA paradigm and its idea of logical separation of concerns to define a service provisioning framework in the homecare domain. The framework will be used in Chapter 3 to compare the existing homecare service provisioning platforms, their functionalities and the challenges addressed by them.

This chapter is organized as follows: Section 2.1 elaborates the concept of dynamic service provisioning, different types of contextual changes and how these changes are addressed in a dynamic service provisioning approach. Section 2.2 introduces a generic dynamic service provisioning platform and its stakeholders. Section 2.3 explains how the dynamic service provisioning and tailoring platforms communicate to create or to tailor an application. Section 2.4 specializes the generic dynamic service provisioning platform (introduced in Section 2.2) in the homecare domain as a

DHSP platform to provide a homecare system. Section 2.5 introduces our service provisioning framework in the homecare domain to explain different challenges and functionalities of a homecare service provisioning platform.

2.1 Dynamic Service Provisioning

In a service-oriented architecture, a *composite application*, is composed of application services provided by possibly different, economically independent service providers. An *application service* is a concrete service that provides a set of functionalities which can be accessed by standard communication protocols such as SOAP. These application services provide access to software or hardware components through which a composite application employs the functionalities of these components. For instance, a medicine dispenser application service provides access to a medicine dispenser device and its functionalities either to dispense a medicine or to know when a particular medicine is taken. Furthermore, a reminder application service provides access to a software component running on an end-user's PDA to show a textual or audio reminder. In a composite application, application services are composed and configured to meet requirements of the application.

In *dynamic service provisioning*, composite applications can update their behaviours with respect to relevant *contextual changes* without or with minimum manpower.

Dey defines context as "*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*" [43, 44]. Some of the changes in this information are considered *relevant contextual changes*, i.e., if they occur, they make the current behaviour of a composite application undesired for the end-users of that application. As such, the composite application should adapt its behaviour by selecting different application services or configuring and composing the selected application services differently. For instance, it should select different medicine dispenser devices (e.g., based on their locations) or configuring a reminder service to play audio message instead of showing textual message or use a reminder service before the medicine dispenser service or vice versa through different compositions.

Due to the complexity of today's software-intensive systems and the high degree of uncertainty in business environments, it is not always feasible to foresee all contextual changes and to plan the corresponding

automated adaptation. Thus, at a later phase in the lifecycle of a composite application, a programmer or an end-user should adapt the behaviour of the application manually. To define which types of contextual changes (from now on, we call them changes) should be addressed by which entity, first we need to classify the contextual changes.

2.1.1 Types of Contextual Changes

We categorized contextual changes as follows:

(a) *Observable vs. non-observable changes*: Observable changes are contextual changes which can be monitored through physical sensors (non-symbolic interface). For instance: location, time, availability (of service or end-user) and vital-signs (e.g., blood pressure value). These changes could be either a new value for a contextual parameter or a new contextual parameter itself. For instance, when a care-receiver takes medication at 3 AM instead of 8 AM, this a contextual change with new value.
Non-observable changes are any contextual changes that can not be monitored through the current physical sensors. For instance, changes in end-users' requirement (goal), development of visual, mental, cognitive, or hearing impairment, and changes in end-users' psychological situation. With the emergence of new technology and sensors, some of the non-observable changes might become observable.

(b) *Foreseen vs. unforeseen changes*: Foreseen changes are contextual changes which are known by the application programmer at design time, also known as a prior. The programmer can define a response of an application to these contextual changes by when defining the application logic. This determines how the application should behave with respect to these foreseen changes. For instance, the programmer knows that a care-receiver's location might change from inside to the outside home, and thus, the programmer defines that the reminder message should be shown on a PDA or a Tablet PC of the care-receiver.
Unforeseen changes are contextual changes which are not known by application programmer at design time. For instance, the programmer might not know at the design time that the care-receiver might measure his blood pressure much earlier than the scheduled time. Therefore, the application is designed in a way that it only checks the vital-sign values at the scheduled time and it could lead to a late application response. Therefore, at a later phase in the lifecycle of the application, the programmer should manually modify the application logic to check the measured vital-

signs whenever it is measured.

2.1.2 Adaptivity, Tailorability and Evolvability

In dynamic service provisioning, a composite application can be reconfigured. This can happen automatically on-the-fly by the system (adaptive service provisioning), by end-users (tailorable service provisioning) or by a programmer (evolvable service provisioning).

The adaptive reconfiguration must be foreseen at design time, and the composite application must be able to monitor relevant changes and react to them at runtime based on predefined application logic. The tailored reconfiguration implies that the end-user recognizes the condition for change (i.e., not observable by the application), but the possible responses to this change have been predefined (i.e., foreseen changes) and can be configured by the end-user. In contrast, unforeseen changes are not known at design time and no possible responses have been planned at design time. Thus, at a later phase in the lifecycle, in the evolvable reconfiguration, a programmer of a composite application must modify the application logic to deal with unforeseen changes. Table *2-1* represents the three aforementioned types of dynamicity.

Table 2-1 : Three types of dynamicity

	Foreseen changes		Unforeseen changes	
	observable	Non-observable	observable	Non-observable
Application	Adaptive	-	-	-
End-user	-	Tailorable	Tailorable (partially)	
Programmer	-	-	Evolvable	

An application is adaptive if it monitors foreseen observable changes and reacts to them based on predefined application logic (*adaptive application*). As an example in the homecare domain, the platform supports an application to monitor the blood pressure of an elderly, i.e., the care-receiver, and then to adapt its behaviour by stopping the current activity and sending an alert to nurses, if the blood pressure is either too high or too low.

An application is tailorable if it can not observe and thus can not monitor changes (this is done by the end-user), but can adapt its behaviour based on reconfiguration by the end-user (*tailorable application*). For example, if hearing impairment of an elderly develops over the execution time, the nurse can increase the default volume of audio reminder to remind the elderly to measure his blood pressure.

An application is evolvable if it facilitates the manual update of application logic (i.e., reducing required manpower and system resources) to address unforeseen changes (*evolvable application*). For example, after introducing the system, if a care-receiver measures his blood pressure much earlier than expected (for instance, at 3 AM instead of its scheduled time at 8 AM), and the blood pressure measurement value is either too high or too low, then the application must send the alert immediately (at 3 AM and not at 8 AM). To address this unforeseen change, the programmer of the application should adapt the application logic.

Although some contextual changes are not foreseen at design time, but can be addressed by the-end users using the available tailoring reconfigurations. For example, the tailoring reconfigurations enable the care-giver to increase the time interval of sending reminder to care-receivers with movement disabilities. At runtime, the care-receivers might go more frequently outside their apartments (as an unforeseen behaviour change), which takes time longer to react to the reminders. To address this unforeseen change, the care-giver uses the existing reconfiguration to adapt the application by giving care-receivers more time to go back to their apartments and to measure their blood pressure. This is possible because the care-giver gets familiar with the system, its reconfigurations possibilities and the care-receivers' behaviour. Therefore, some unforeseen changes might be partially addressed by tailorability. However, this is not always possible and thus for some other unforeseen changes, the application logic must be manually updated by the programmer.

2.2 The Provisioning Platform and its Stakeholders

We define a *dynamic service provisioning platform*, as an adaptive, tailorable and evolvable application service provisioning platform. The platform should support applications for adaptivity, end-users for the tailorability of the applications and programmers for the evolvability of the applications

In our platform, application logic can be defined by a service plan. A *service plan* consists of one or more service building blocks and describes the configuration and composition of instances of these service building blocks with respect to run-time circumstances. A *SBB*, in turn, defines a set of functionalities in the abstract level that can be implemented by alternative application services. For instance, a SBB of medicine dispenser can be implemented by different medicine dispenser devices which might be provided by different vendors. The service

plan specifies the behaviour of a composite application at runtime by specifying which component service will be selected for a SBB and how the selected service will be configured and orchestrated.

In our dynamic service provisioning system, as illustrated in Fig. *2-1*, a service provisioning platform composes different types of application services: infrastructure services (provided by the platform), application services (provided by third-parties) and internal application services (provided by the platform). The *Infrastructure services* are application scenario-independent, while application services are used for a specific application scenario. Besides, the infrastructure services are used to select, configure and compose the application services.

In addition, there are several stakeholders, for example programmers, domain experts and end-users. In our case, domain experts are nurses, or more generally care-givers. To decouple the concerns, we assume that there is a separate *tailoring platform* that takes care of the service plan creation and tailoring, and eventually deploys the service plan to the provisioning platform for execution.

The *end-users* use the composite applications that run on top of the platform. In our case, they are people needing homecare applications, such as patients or some elderly people, generally called care-receivers in this thesis. End-users interact with the applications either directly with the platform through its internal application services, or through the application services provided by third-party providers. Nevertheless, the end-users are not aware of these different service providers and use all the services as provided by the platform.

A domain-expert can be an end-user too, if the application is supposed to interact with the end-user during its execution. This decentralized architecture is not restricted to homecare systems and we regard it as representative of all decentralized service-oriented systems.

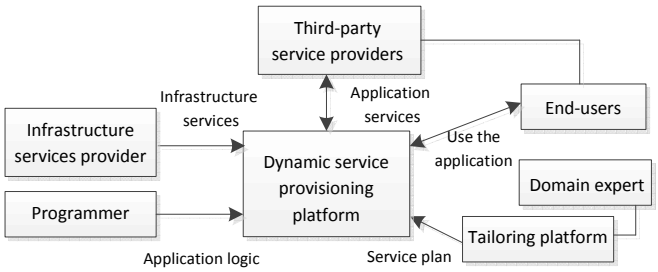

Figure 2-1 : Actors in a dynamic service provisioning system

The *domain experts* can define the behaviour of the application by assigning values to the configuration parameters of the service plan. In response, the composite application updates its behaviour based on

its service plan. However, addressing unforeseen changes might need new configuration parameters, values or even new orchestrations. This requires IT-knowledge that the domain expert usually does not have. In this case, we assume that the *programmer* who can do arbitrary IT-specific tasks to define or modify the application logic. We assume that the programmer acquires the requirements from the domain expert and accordingly updates the application logic. The programmer must also update the tailoring platform to enable the domain expert to use the new or modified service plan.

2.3 Provisioning Platform vs. Tailoring Platform

To adapt the homecare applications to contextual changes, the provisioning platform should be able to adapt applications while they are being executed. If the runtime adaptation is not applicable, the care-givers modify the homecare services at design time through the tailoring platform and then (re)deploy them to the provisioning platform. The care-givers are either professional care-givers (e.g., nurses, doctors) or social care-givers (e.g., friends and family members). In order to avoid any misunderstanding, we should explain our perception from the tailoring and provisioning platforms and their interaction with each other. Figure *2-2*, shows the overall view of the tailoring and provisioning platforms.

We define service tailoring as a process which consist of all activities that a care-giver performs prior to the provisioning of homecare applications. The service tailoring ends with the specification of the homecare applications, i.e., the service plan that constrains the behaviour of the homecare services at runtime. The service plan should determine the corresponding desired behaviour of the applications with respect to observable foreseen changes. The tailoring platform is responsible to enhance the creation and tailoring of the service plans by providing GUI for the care-givers. After the service plan is confirmed by the care-givers, it is deployed to the provisioning platform to be executed.

In our definition, a service plan refers to one or more *SBBs* and describes the configuration and orchestration of instances of these SBBs as well as decision rules with respect to run-time behaviour. The SBBs, like a medicine dispenser or reminder, are the smallest manageable services which cannot be further broken down into smaller services from the care-givers point of view. Configuration parameters allow the care-givers to specify different aspects of the SBBs such as service operations and user interface modalities. Orchestration schemes

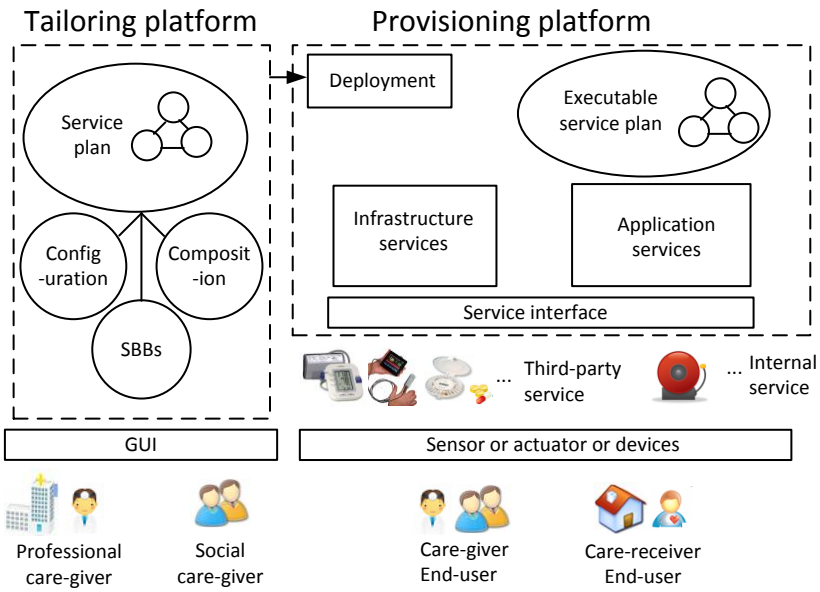

Figure 2-2 : The tailoring platform and the provisioning platform

determine how SBBs are composed. Decision rules determine the possible adaptation at runtime, based on evaluation of the rules with runtime data (e.g., context values). For example, decision rules can be used to choose between alternative operations of one SBB or between alternative data and control flows among the SBBs, based on specific runtime circumstances.

Since we aim to support dynamic service provisioning, we follow the late binding mechanism which is more adaptable when compared with early binding [133]. The late binding abstracts concrete details of SBBs from the care-givers and enables them to create and tailor the service plan in a straightforward way. Later, in the provisioning platform, the SBBs of the service plans are bound to the *application services*. Hence, the service plan should be detailed enough to enable the provisioning platform to convert them to an executable format. As such, the provisioning platform checks if the care-givers give values to all the configuration parameters of a service plan before the deployment of that service plan.

By service provisioning, we mean the execution of the service plan based on functionality offered by available services at runtime. The provisioning platform should bind the abstract SBBs used in the service plan to the *application services*. The late binding and execution of the *application services* based on the service plans are supported by *infrastructure services* like orchestration and context services,

which are provided by the provisioning platform. The *application services* are provided either by the third-party service providers or the internal services provided by the provisioning platform provider. For example, based on our application scenario, oxygen saturation measurement is provided as a third-party service which is accessible for other services through the Internet. An example of internal service is an alert service which is implemented by the provisioning platform. The provisioning platform defines a set of service interfaces for all types of application services to enable them to communicate with the provisioning platform.

2.4 DHSP platform in a Homecare System

A *homecare system* includes platforms (both tailoring and provisioning platforms), application services (both third-party and internal), devices, data and networks that are required to support independent living of care-receivers. We define *care-receiver* as an elderly person who lives in a care home and receives care services from their care-givers. A *care home* is either a private home located outside of a care center, or a unit located inside a care center. *Care-givers* are experts and volunteers who provide care and social services to elderly. As such, the provisioning platform interacts with care-receivers and needs to be installed per care home to execute its own services. In contrast, the tailoring platform interacts with the care-givers and one tailoring platform can be employed for a care center of several care homes.

As part of U-Care (User-tailored Care services platform) project [142], we developed a prototype of a homecare system. Figure *2-3* shows the U-Care system as an instance of the homecare system.

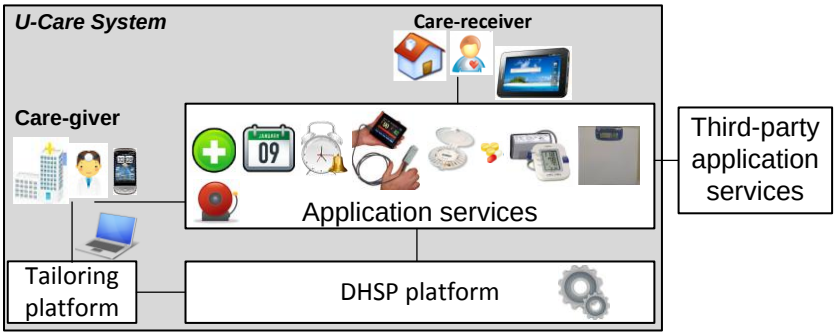

Figure 2-3 : The U-Care system as an instance of the homecare system

To address the dynamicity challenges, we define a *Dynamic*

Homecare Service Provisioning (DHSP) platform as an dynamic service provisioning platform to address the dynamicity requirements of the homecare domain including the required SBBs for homecare applications. These SBBs represent a set of application services (both third-party and internal) which are required for our application scenarios. According to our application scenario, we have 7 third-party application services: weight measurement, blood pressure measurement, oxygen saturation measurement, medicine dispenser, calendar, reminder and reporting service, and one internal application service which is the alert service. These application services, their SBBs and functionalities will be elaborated in Chapter 5.

2.5 Homecare Service Provisioning Framework

In this section, we introduce a functionality layering framework to analyze the challenges to be addressed by a homecare service provisioning platform including both service realization and composition. This framework is useful for both the service developers as well as the researchers. Developers can systematically breakdown the problems in different layers and find the corresponding solutions whereas the researchers can use the framework to find issues in the existing solutions. We also discuss what interoperability challenges exist between different functional layers. This framework helps developers and domain experts identify interoperability challenges that exist in homecare systems provisioning. To the best of our knowledge, the proposed framework is the first one that attempts to classify service realization and composition issues in the homecare domain. A service composition framework introduced in [124] assumes the availability of services and focuses on dynamic service composition. In the homecare service provisioning, availability of the homecare services is not a valid assumption. Due to several low level hardware devices which might be employed by the homecare applications, service realization challenges should also be taken into account.

2.5.1 Interoperability Issues

The complexity of interoperability and its multifaceted characteristics led to several definitions [92]. The IEEE Glossary defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [76]. System can be understood in terms of different perspectives such as information and behaviour. The interoperability

challenges can be considered based on these different perspectives of systems. The interoperability challenges from the *information* perspective can be seen as the difficulty in sharing information between systems due to differences in coding, formatting and data representation schemes. An existing homecare platform called SAPHIRE employs a cross-enterprise document sharing architecture to tackle information interoperability like discovery of and access to relevant electronic health records [112].

The interoperability challenges from the *behaviour* perspective can be seen as the difficulty in using each others functions due to differences in their protocols. Another existing homecare platform, MPOWER, introduces a solution based on SOA and a message translator pattern to provide interoperability services for the Ambient Assisted Living (AAL) environment [104]. To achieve interoperability, distribution, heterogeneity and dynamicity [26] should be addressed at design and implementation time. We explain these issues and their causes in homecare domain as follows:

– *Distribution:* In distributed systems, a reliable medium for communication and seamless addressing is considered as an interoperability challenge. The medium should provide location transparency and hide the complexity of the underlying communication infrastructure. Care-receivers and care-givers should be able to use homecare applications and their application services which are located in different places, for instance, different rooms, through the provisioning platform. Care-givers may use homecare applications from both inside and outside care home. Third-party application services should be accessible through the Internet by the homecare applications.

– *Heterogeneity:* Interoperability is especially hard to achieve if distributed systems are made up of heterogeneous subsystems that are developed, managed and owned by different organizations. From the software point of view, the applications might follow different protocols for discovery (e.g., WS-Discovery and SLP), interaction (e.g., SOAP and RMI) and transportation (e.g. TCP and UDP). From a hardware point of view, they can have different capabilities, varying from a resource-constrained sensor to a computing device with high computation resources. The software and hardware heterogeneity bring us a set of constraints which should be taken into account at implementation time.

– *Dynamicity:* We define dynamicity in homecare domain as

any relevant contextual changes that make current behaviour of homecare applications not suitable for the end-users. The contextual changes coming from the care-receiver, homecare applications and environment characteristics [100]. Some of the dynamicity like a visual impairment development occur over a long period of time and can be handled by a care-giver (tailorability) or a programmer (evolvability). On the other hand, some other contextual changes like changes in location and activity might occur in short-term and must be handled in runtime by the platform (adaptivity).

2.5.2 Methodology

The functional aspect of homecare applications is the focus of the work presented in this thesis. Moreover, we focus on the interoperability challenges from the behaviour perspective. We consider the behaviour perspective because it allows us to look at the applications behaviour while using each others' functions. It is difficult to relate the information interoperability to the functional layers as defined in our framework. Therefore, we do not consider information interoperability challenges in this framework.

In order to classify challenges and break them down into smaller ones, we employ the SOA functional layering as defined in [18], as the basis for defining our framework. The service-oriented architecture provides guidelines to achieve interoperability that assist different organizations to be dynamically integrated regardless of their technologies, platform and application specification. The SOA paradigm can be considered as a set of distinct layers of functionalities which describes a logical separation of concerns [19, 117]. Each layer employs the functionalities of its lower layer and provides new functionalities for its next higher layer.

Based on the work presented in [18], we defined our framework as a set of five functional layers. These layers are numbered 1 to 5 and cover different aspects, of a homecare domain, spanning from low level hardware devices to high level end-user applications. The lowest layer is numbered 1 and the highest layer is numbered 5 as shown in Figure *2-4*. The layers are defined such that each layer employs the functionalities of its lower layer and provides new functionalities to its next higher layer. Given two layers of the proposed framework, a layer is considered lower than the other if it covers more concrete entities of the homecare domain. This means that the lower layers are mostly concerned about service realization while the higher layers are mostly concerned about service composition. This kind of layering is useful to analyze problems in the homecare domain in a systematic way.

In the proposed framework, physical devices like sensors are located in the lowest layer which is numbered 1. The software components which realize service interfaces to communicate with the physical devices are placed on the next layer which is numbered 2. The software components which provide a uniform service layer on top of all the existing service interfaces to hide the heterogeneity and distribution of underlying components are placed in the layer which is numbered 3. A set of basic functionalities (SBBs) which can be composed and configured by the care-givers to create homecare applications are placed in the layer which is numbered 4. This layer hides the details of concrete implementations of the lower layers. The highest layer, which is numbered 5, contains the homecare applications which directly interact with care-receivers.

2.5.3 Functionality of the Layers

Our proposed homecare service realization and composition framework is shown in Figure *2-4*. The left part shows the functional layers which are provided by homecare applications. The right part presents a platform which provides several services like orchestration and discovery services to the application functional layers. The platform enhances the development and execution of the homecare applications. We aim to emphasize on service realization and composition challenges from the application point of view, to see how the platform part should address them at different levels. For example, communication relationships among geographically distributed systems are established through the platform. Composition relationships among layers are shown by a line ended with a circle, and can have 1:n and 1:1 cardinality.

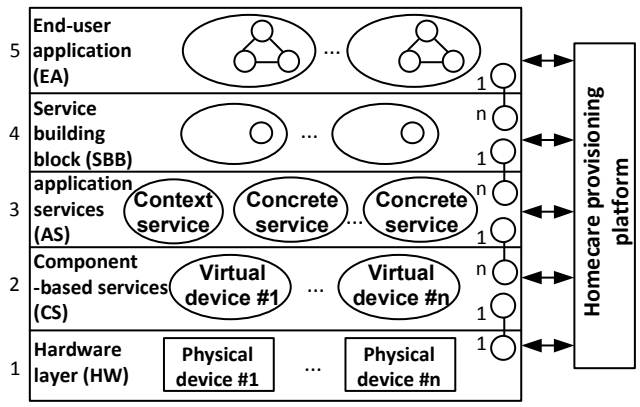

Figure 2-4 : The functional layers of the homecare applications

Figure *2-5* shows an instance of the functional layers of the

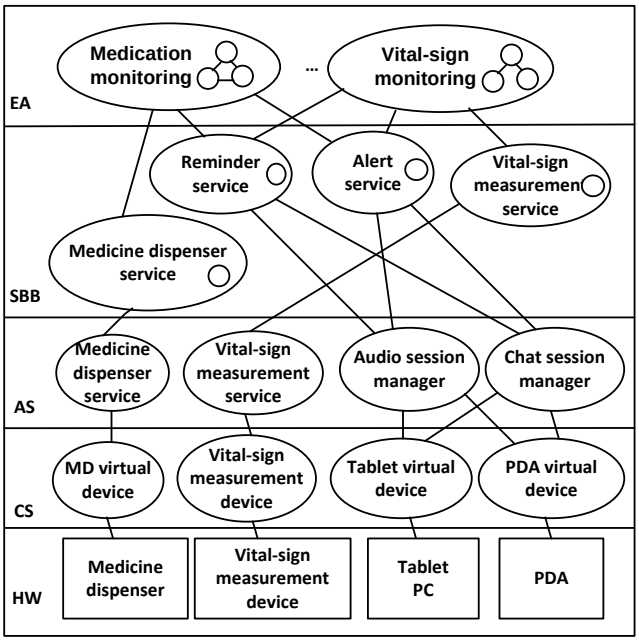

Figure 2-5 : An instance of the functional layers of the homecare applications

homecare applications according to our application scenario explained in Section 1.2. The lines between layers show the composition relationships which comply with the cardinality constraints between layers shown in Figure *2-4*.

Hardware (HW): in this layer, we have three types of physical devices: *actuator*, *sensor* and *multi-media interface*. For the target scenarios, we can mention medicine dispenser as an actuator, vital-sign measurement device as a sensor (e.g., blood pressure measurement), and Tablet PC and PDA as multi-media interfaces. The constituents of the *HW* layer can only provide their functionalities to their representative components located in the next higher layer. Therefore, any interaction with the devices located in the *HW* layer, should be done through their corresponding *virtual devices* implemented in the *Component-based services (CS)* layer. The composition relationship between *HW* and *CS* layer has 1:1 cardinality. It means that for each physical device in the *HW* layer, there is only one representative *Virtual device* in the *CS* layer.

Component-based services (CS): in this layer, we have a set of *virtual devices* which are service realizations of the physical devices in

the lower layer. In general, the *virtual devices* realize the services by three main strategies: wrapper, mediator and generic service interface. As a wrapper, a software component encapsulates a physical device and provides its own service interface, like an OSGi bundle [65]. As a mediator, a software component or an application, plays an intermediary role between the physical devices and the upper functional layer. For instance, a component so-called frame sensor adapter (FSA) allows to uniformly collect data from sensors [95]. As a generic service interface, a *virtual device* provides a set of generic interfaces to call a physical device. For the target scenarios, a *MD (medicine dispenser) virtual device* provides access to a web portal for interacting with the physical *Medicine dispenser*.

Application service (AS): in this layer, we have the application services that provide the functionalities of the *virtual devices* as a uniform service layer. We have two types of application services: *Context* and *Concrete services*. *Context* application services aggregate raw data from one or several *virtual devices* to infer whether to trigger a particular event or not. Accordingly, the composition relationship between the *IS* and *CS* layers has 1:n cardinality. We define the *Concrete services* to hide the heterogeneity and distribution of the *virtual devices* and to provide a uniform protocol to discover and interact with them. *Concrete services* have two main functionalities: adapter and messaging. As adapters, they provide interoperability between two or several *virtual devices* which might use different discovery and interaction protocols. The messaging functionality provides transparent communication with the *virtual devices*. These two functionalities are supported by the platform services. For the target scenarios, the *Vital-sign measurement service* is a *Context* service to provide vital-sign information and the *Medicine dispenser* is a *Concrete service* provides a uniform access for the *MD (medicine dispenser) virtual device*.

Service building block (SBB): in this layer, we place homecare related *SBBs*, i.e., the smallest manageable services, which cannot be further broken down into smaller services from the care-givers' point of view. However, from the platform's point of view, these services might exploit one or several of the concrete and context services which are provided by the *AS* layer. For instance, the *Reminder service* employs the *Audio session manager* to send audio messages and *Chat session manager* to send textual messages. Therefore, the *SBB* layer has 1:n composition relationship with the *AS* layer. In our application scenarios, from the care-givers' point of view, there are several *SBB*s

like *MD* and *Reminder*. *SBB*s are reusable and context-independent basic services which can be composed to create and tailor homecare applications. Each of the *SBB*s can be seen as a service interface with a set of operations. In addition, homecare applications need a set of events to implement their logic. Some SBBs might have notification services to handle these events. These notifications are defined based on one or several events triggered by the services located in the *AS* layer. For instance, a *vital-sign measurement service* has a notification to indicate that a new vital-sign measurement is received, in which case the *Vital-sign monitoring* application stops sending reminders.

End-user application (EA): in this layer, we place the homecare applications which are represented by a set of service plans that fulfill the homecare applications objectives. For instance, the objective of the *Medication monitoring* application is to remind the care-receiver to take medicine on time. The service plan specifies the application behaviour at runtime and can be defined by composing services. Each homecare application has one or several service plans, which consist of one or several *SBB*s. Therefore, the composition relationship between the *EAS* and the *SBB* layer has 1:n cardinality. We classify the functionality of this layer in three steps: generation, evaluation and transformation. In the first step, a service plan is generated by the care-givers based on the service interfaces of the available *SBB*s. In the second step, the service plan should accommodate the care-receiver's preferences and the runtime contextual environment, to prioritize the alternative possibilities of service execution. In the third step, the service plan must be transformed to an executable composed service by mapping the service interfaces to concrete service providers, like an available medicine dispenser.

2.5.4 Vertical Interoperability Challenges

As mentioned in Section 2.5.2, we focus on the interoperability challenges of systems from the behaviour perspective in the homecare domain. We associate the interoperability challenges with the composition relationships between the functional layers as defined in our framework. These challenges, which are categorized into the three homecare-related interoperability issues (explained in Section 2.5.1), are shown in Table *2-2* and discussed as follows:

HW-CS: the composition relationship between these two layers is one-to-one. This means that from the *AS* layer there is no distinction between physical devices and their corresponding virtual

devices. Therefore, we do not consider any dynamicity and distribution challenges in this composition relationship. However, we consider the heterogeneity challenge because the hardware devices could come from different manufacturers and with different proprietary solutions. We identify three challenges with respect to device heterogeneity:

– *Behaviour*: providing a uniform service interface for the physical devices despite their heterogeneous behaviour is a tough challenge. The uniform service interface should support both static and non-static characteristics of the behaviour of the virtual devices. Different virtual devices, for example, can have the same static characteristics like service interface signature with different non-static characteristics like various orders of function access.

– *Resource-constrained devices*: having various physical devices with resource limitations (e.g., memory size, computation power) leads to several constraints on their usage.

– *Communication*: physical devices could have different communication protocols which the virtual devices should support to provide communication protocol transparency or even location transparency.

CS-AS: the composition relationship between these two layers is many-to-one. This means that one application service could communicate with one or more component-based services. Therefore, we consider dynamicity, heterogeneity and distribution challenges in this composition relationship. We identify two challenges with respect to dynamicity:

– *Life-cycle of virtual devices*: providing a cooperation environment is required to enable several application service providers to manage their component life-cycle (for example, install and configure each component) without interfering other components' operations.

– *Plug-and-Play support*: availability of the virtual devices in the homecare environment is not guaranteed. The virtual devices should be allowed to register themselves to the system automatically and be operational as soon as they become available.

We identify one challenge with respect to heterogeneity:

Table 2-2 : The summary of vertical interoperability challenges (X : applicable)

	Interoperability challenges	Dy	Hg	Ds
SBB-EA	Tailorability	X		
	Context-aware composition	X		
	Context-aware configuration	X		
IS-SBB	Generic SBB model	X		
	Context-aware selection	X		
	Late binding	X		
AS-IS	Life-cycle of virtual devices	X		
	Plug-and-play availability	X		
	Discovery and interaction protocols		X	
	Location transparency			X
HW-CS	Behaviour		X	
	Resource-constrained devices		X	
	Communication		X	

– *Discovery and interaction protocols*: the definition of the CS layer implies that there should be no heterogeneity issue above that layer. However, some strategies used by the CS layer, like the generic service interface, only address the heterogeneous behaviour of the virtual devices. Therefore, there is a need to support different discovery mechanisms (e.g., WS-Discovery, SLP) and interaction protocols (e.g., SOAP, RMI) between the CS and AS layers.

We identify one challenge with respect to distribution:

– *Location transparency*: component-based services are physically distributed. In that case, providing a location transparent communication infrastructure is a challenge.

AS-SBB: the composition relationship between these two layers is many-to-one. This means that one service building block could communicate with one or more AS services. We assume that the AS layer hides all issues related to distribution and heterogeneity. Therefore, we consider only the dynamicity challenges in this composition relationship. We identify three challenges with respect to dynamicity:

– *Generic SBB model*: providing a generic mechanism to model the SBBs, understandable for both the platforms (provisioning and tailoring) and care-givers, is considered as a challenge, specially due to complexity of the behaviour of the services provided by the lower layers, and their runtime contextual situations which are not known at design time. SBBs should have two representations: internal and external. The internal representation is used to be

interpreted by the platforms while the external representation is used to present the *SBB*s to the care-givers.

– *Context-aware selection*: providing contextual information about care-receivers which is needed to select one of the alternative application services for one SBB. For instance, the care-receiver is out of his care home and thus the chat session manager service running in the PDA of care-receiver must be selected for the reminder SBB.

– *Late binding*: to support *Context-aware selection*, the platform should be able to bind the SBBs to the application services at runtime [133].

SBB-EA: the composition relationship between these two layers is many-to-one. This means that one end-user application could be composed by one or more *SBB*s. In this composition relationship, we consider only the dynamicity challenges but not the heterogeneity and distribution challenges because of the same reason as mentioned in *AS-SBB* composition relationship. We identify three challenges with respect to dynamicity:

– *Tailorability*: for some contextual changes, the possible responses to these changes have been predefined and can be configured by the end-user, such as care-giver. For example, in case of hearing impairment development of a care-receiver, the care-giver can increase the volume of audio reminder. Therefore, the care-giver should be able to reconfigure a running homecare application without interrupting other running applications. This reconfiguration must be done within a specific time without systems errors.

– *Context-aware composition*: based on the runtime contextual information, the selected application services can be composed differently. For instance, a reminder must be sent before a vital-sign measurement service to remind a care-receiver to take a medication or reminder must be sent after measuring the vital-signs.

– *Context-aware configuration*: based on the runtime contextual information, an application service selected for a specific SBB could be configured differently. For example, the selected audio reminder service could have different voice volume levels during the day and over the night.

Chapter 3

Related Work

"No matter how busy you may think you are, you must find time for reading, or surrender yourself to self-chosen ignorance."
— Atwood H. Townsend

This chapter positions our concepts with respect to related existing terminology and provides an overall view of existing dynamic service provisioning approaches. First, we study the existing homecare systems using the framework introduced in Section 2.5. We observe that the dynamicity challenges have been not addressed by the existing homecare systems as much as other challenges. This motivates us to zoom into the dynamicity challenges and design a DHSP platform to address them. Then, we study the related work out of the homecare domain which could be applicable to address the dynamicity challenges. Several research areas, such as context-aware applications, service discovery, dynamic configuration of component-based application and hybrid service composition can be considered as our related work. However, with respect to the requirements which are discussed in Chapter 4, we limit our related work to hybrid service composition approaches that use a combination of processes and rules. We find the combination of processes and rules as the most appropriate method to address the dynamicity challenges in the homecare domain [10] for instance, to be compliant with rule-based and care-flow medical protocols.

This chapter is organized as follows: Section 3.1 studies existing homecare systems using the framework introduced in Section 2.5 and motivates our decision to zoom into the dynamicity challenges. Section 3.2 positions our concepts with respect to related existing terminology and provides an overall view of existing dynamic service provisioning approaches. Section 3.3 studies existing hybrid service composition approaches that use a combination of processes and rules.

3.1 Studied Homecare Systems

There is an emerging trend in industrialized countries for using IT-based homecare services and thus several promising homecare system prototypes have been developed in academia [98, 110, 21, 55, 170, 91, 154, 37, 51, 139, 47, 15]. We classify the homecare systems in the following categories:

– Patient Record Systems: Homecare systems as Information systems should be able to share the information of a care- receiver such as vital-signs with other healthcare-related applications, such as the electronic medical patient record application of a hospital [70]. Thus several homecare systems emphasize the information interoperability and knowledge-based representation of care-receivers' data [21, 51, 126]. The Health Level Seven (HL7) standard provides some guidelines on how to facilitate knowledge sharing among different healthcare applications [23].

– Monitoring Systems: Homecare systems should notify care-givers when hazardous situations are detected involving care-receivers by utilizing sensors which a care home is equipped with [98, 110, 47]. For instance, an alert must be sent to a care-giver if the blood pressure value of a care-receiver is abnormal. Moreover, care-givers have access to the history of monitoring records (e.g., blood pressure measurement records) to prescribe appropriate treatments and take proactive actions to maintain the care-receives' health condition.

– Reminder Systems: Forgetfulness is one of the common problems of elderly people. Due to this, elderly people might forget the scheduled time for their necessary health activities, such as taking medicine or measuring their own vital signs. Homecare systems should remind care-receivers of a specific activity, preferably using various modalities such as textual, audio and video reminder messages [98, 110, 47].

– Social Interactive Systems: Feeling lonely is a common issue among elderly care-receivers. Due to this, homecare systems should support audiovisual communication between care-receivers and care-givers, including social care-givers like fiends and family, or other with care-receivers [15, 139]. This enables care-receivers (a) to discuss care related issues with their care-givers, and (2) to diminish their loneliness by setting-up and participating in social

activities with other care-receivers.

Our research scope is limited to the provisioning part of the homecare systems, which includes either one or all the three types of systems: monitoring, reminder or social interactive systems. As we discussed in Section 2.5, we emphasize behavioural interoperability instead of information interoperability which is the purpose of patient record systems. Because, we have only one care center in our application scenario that does not interact with other healthcare organizations such as a hospital and thus information interoperability is not covered in this work.

3.1.1 Homecare Provisioning Platforms

We observed that two middleware technologies are well exploited by homecare provisioning platforms: OSGi [12] and web services [66]. To cover a wide range of solutions in SOA-based homecare applications, we have studied academic homecare projects that employ both aforementioned technologies. We have selected three of these projects as examples to show their technologies and solutions. We have used the service provisioning framework, introduced in Section 2.5 to explain these projects and how they address the identified challenges. They use discrete heuristics in each of the layers of our framework.

The three selected projects are: (1) *MATCH (Mobilising Advanced Technologies for Care at Home)* [98], (2) *MPOWER (Middleware Platform for eMPOWERing cognitive disabled and elderly)* [110], and (3) *Amigo (Ambient Intelligence for the networked home environment)* [15].

Table *3-1* summarizes the technologies and solutions employed by these projects in each of the functional layers of our framework. Moreover, Table *3-2* shows to what extent these projects support the interoperability challenges identified for each of the functional layers. Below, we explain both tables by discussing each layer:

Table 3-1 The technologies and solutions are employed by projects 1-3

	Match(1)	MPOWER(2)	Amigo(3)
EA	XML, Directed graph Interactive evaluation	BPEL, UML service models	BPEL, OWL-S Online semantic reasoning
SBB	XML, Interaction model OWL-based events	WSDL, IBM UML Profile	OWL-S, Amigo-S Rule-based notification
AS	Message broker component	Open ESB	Middleware SDP and SIP meditation
CS	OSGi, UPnP Wrapper	Platform-independent models Mediator	OSGi, .NET, UPnP Generic service interface
HW	Residential gateway	Proprietary framework (FSA)	Domotic infrastructure

– In the HW layer, $Match$ uses OSGi component, i.e., bundles to

Table 3-2 The challenges addressed by the three projects (Supported:S, Partially supported:P, Not supported:N)

	Challenges	(1)	(2)	(3)
SBB-EAS	Tailorability	N	N	P
	Context-aware configuration	S	N	S
	Context-aware composition	P	N	S
IS-SBB	Generic SBB model	S	S	S
	Context-aware selection	P	N	P
	Late binding	S	P	S
CS-AS	Life-cycle of virtual Dev.	S	N	S
	Plug-and-play availability	S	N	S
	Discovery and interaction	S	S	S
	Location transparency	S	S	S
HW-CS	Behaviour	S	S	S
	Resource-constrained Dev.	S	S	S
	Communication	S	S	S

represent physical devices. With respect to *Resource-constrained devices*, these bundles are located in a residential gateway (i.e., a server in care home) or any intermediary which has Java Virtual Machine (JVM) installed and can communicate with their corresponding devices through heterogeneous *Communication* protocols.
MPOWER exploits a proprietary framework, so-called Frame Sensors Adapter (FSA) [95], to collect data from several sensors. *MPOWER* delegates the management of heterogeneous *Communication* protocols and *Resource-constrained devices* to FSA.
Amigo has developed a domotic infrastructure to decouple the technologies used in the *HW* layer from the virtual devices in the *CS* layer. The infrastructure supports heterogeneous *Communication* protocols and is implemented in C which is suitable for dealing with *Resource-constrained devices*.

– In the *CS* layer, *Match* uses OSGi to manage the dynamicity of the *Life-cycle of virtual devices*. It utilizes the UPnP protocol to address *Plug-and-play support* by providing simplified installation upon Internet-based communication protocols. The OSGi bundles address the heterogeneity of the *Behaviour* of physical devices by using wrapping techniques.
MPOWER employs a platform-independent model which can be realized on several platforms such as web services, .NET, CORBA and J2EE. We could not find explicit information with respect to the support for *Life-cycle of virtual devices* and *Plug-and-Play support* in the literature concerning *MPOWER*. *MPOWER* uses

the FSA as an adapter to hide the heterogeneous *behaviour* of the physical devices.
Amigo developed a middleware which can be installed on individual platforms to support both OSGi and .Net standards. It also follows OSGi and UPnP based techniques as used in *Match* to support the *Life-cycle of virtual devices* and *Plug-and-play support*. *Amigo* has developed a domotic service model specification to address the heterogeneity of the *Behaviour* of physical devices. The virtual devices can use the generic service interface instead of directly communicating with the heterogeneous physical devices.

– In the *AS* layer, *Match* has developed its own message broker for the messaging functionality of the *Concrete services*. This broker is implemented as an OSGi bundle to address *Late binding* and *Location transparency*. As *Match* has employed wrapping in the *CS* layer, there is no need for adapters and the heterogeneity of *Discovery and interaction protocols* is already supported by the *CS* layer.
MPOWER has defined an enterprise service bus based on Open ESB for messaging. It supports *Location transparency* but it does not completely support *Late binding* (for example, it does not support dynamic endpoints). *MPOWER*, like *Match*, also does not need adapters due to the use of mediation in the CS layer.
Amigo has developed an interoperable middleware core that provides a service discovery protocol (SDP) and a service interaction protocol (SIP). The SDP addresses the heterogeneity of *Discovery protocols* by parsing the input protocol and composing the target output protocol. The SDP dynamically instantiates a stub from the description and reference of the services. A stub that addresses the heterogeneity of *Interaction protocols* and *Location transparency* can act as an intermediary adapter.

– In the *SBB* layer, *Match* describes the *SBB*s in XML format from an abstract level to concrete interaction interfaces [109]. The abstract level can be used as the external language for the care-givers to support a *Generic SBB model*. This ease of use is provided by a technology-agnostic description of services written in OWL (Web Ontology Languages) to associate a set of predefined events to their possibly corresponding virtual devices, using human-understandable concepts. *SBB*s interface matching [100] supports the dynamic *Context-aware selection*.
MPOWER has described *SBB*s as WSDL files for the internal

purposes. Due to the complexity of WSDL for unskilled users, the IBM UML profile for software services [86] has been exploited as the external language to provide *Generic SBB model*. In the publications on *MPOWER*, we could not find explicit support for a *Context-aware selection*.
Amigo developed the Amigo-S language based on OWL-S to describe services for both external and internal perspectives at different levels of abstraction to provide *Generic SBB model*. Moreover, *Amigo* provides rule-based awareness and notification services (ANS), which can enhance the *Generic SBB model* for the notification services. Amigo-S is part of a comprehensive approach towards semantic service description, discovery, composition, adaption and execution (SD-SDCAE) which specifies QoS as well as the behaviour of services. It uses runtime semantic reasoning to support *Context-aware selection*.

– In the *EAS* layer, *Match* uses a directed graph to define a service plan, by matching the *SBB*s interaction interfaces [100]. It generates several service plans for a target scenario by traversing all possible paths in the graph. The abstract service plan, excluding detailed service interface, supports a *Simple service plan*. The possible multi traversal paths enable the platform to provide *Context-aware compositions* by generating alternative service plans based on available *concrete services* at runtime. In the publications on *Match*, we could not find explicit support for *Tailorability*.
MPOWER has a set of predefined UML service models which are implemented by its own domain-specific modeling language (DSML) and understandable for care-givers. Based on the target scenario and the existing service plan models, it generates the service plan and transforms it to an executable composed service using WS-BPEL. In the publications of *MPOWER*, we could not find explicit support for *Context-aware compositions*, *Context-aware selection* and *Tailorability*.
In *Amigo*, tasks are modeled as abstract workflows based on a semantic contextual model to provide a *Simple service plan*. This semantic contextual model is created with a user-friendly GUI. Several alternative service plans are generated by composing available *Concrete services* in different ways at runtime. Then, the generated service plans are transformed to executable composed services using WS-BPEL. For *Tailorability*, an ontology visualization and environment modeling tool (so-called VantagePoint [148]) has been provided. However, this tool mainly

emphasizes contextual and environment modeling, and does not support the creation of composition (composed applications).

3.1.2 Dynamicity in the Scope of Our Research

Although we selected only three SOA based homecare projects, the proposed framework can be used to analyze other SOA based homecare projects to investigate to what extend the identified challenges are addressed by them. Analysis of larger number of SOA-based homecare platforms would help us define homecare challenges more accurately and refine the proposed framework. If we performed a large scale analysis of homecare platforms we could possibly identify some new challenges or the challenges which are already identified may disappear. Such a large scale analysis, however, would require more research efforts and is left for future work.

Based on our study, we observed that existing solutions can handle distribution and heterogeneity issues. However, dynamicity issues, which affect almost all the functional layers and play an important role in homecare service provisioning, are not well addressed and therefore further research is required. With respect to dynamicity, the service plan should be simple but detailed enough to enable the homecare provisioning platform to select, configure and compose the available concrete services at runtime in a lightweight and context-aware manner.

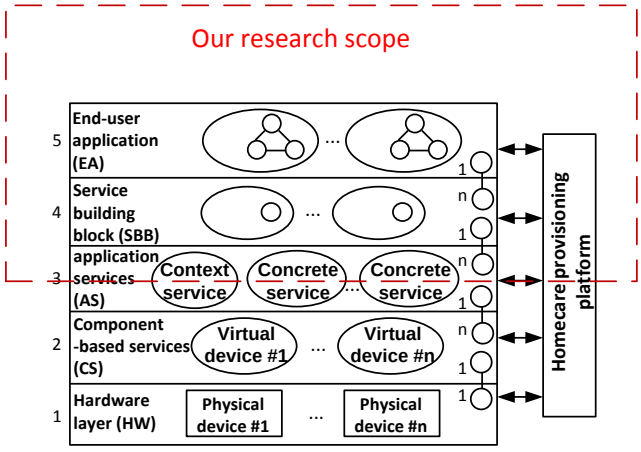

Figure 3-1 : The scope of our research based on the proposed framework (explained in Section 2.5)

With this observation, we dive into dynamicity challenges as the core of our research, specially concerning the tailorability and evolvability. Figure *3-1* shows our research scope. Most of homecare

systems, which we have studied, emphasize the idea that the application itself can adapt its behaviour as we called adaptivity and study how the adaptivity can be improved for instance, by ontological reasoning. Some of the homecare systems consider the tailorability for instance, Amigo [15]. To the best of our knowledge, none of them consider the evolvability and investigate how it can be improved. Moreover, with respect to our application scenarios, we focus on the top three functional layers and the challenges between $AS\text{-}SBB$ and $SBB\text{-}EA$ layers. Because all of our service providers can provide their services over the Internet and we just need to provide a set of application services to adapt their functionalities based on our SBBs. Thus our DHSP platform supports the functionalities of the top three layers.

3.2 Dynamic Service Provisioning

3.2.1 Terminology

As mentioned before, the provisioning platform should be able to adapt the homecare applications with respect to runtime contextual changes. The adaption is done based on the configuration information of SBBs and their orchestration patterns in the service plan. As such, several research fields such automatic (also known as adaptive or dynamic) service composition [123] [31] and dynamic configuration can be considered as the related work. To clarify our terminology and to position it with respect to exiting terminologies, we compare our dynamic service provisioning with the related research areas as follows:

– $Ambient\ and\ Pervasive\ System$ [87]: A homecare system can be considered as an ambient [137] or pervasive system [120]. Because in ambient or pervasive computing the system can be used "$everywhere\ at\ anytime$", and not only on personal computer or laptop [127]. In such homecare system, the focus is how to aggregate contextual information from distributed sensors and accordingly to infer what the system should do. However, based on the scope of our research, application scenarios and service providers we can assume that all the sensors, actuators and devices are accessible through a high-level application services. Thus we focus on selecting, configuring and composing these application services in the top three layers of our framework (explained in Section 2.5), rather than focusing on the service realization in the component-based layer.

– *Context-aware Computation*: In context-aware computation, the application adapts its behaviour with respect to the context of the application including its end-users. Based on a general definition "*a system is context-aware if it uses context to provide relevant information and services to the user, where relevancy depends on the user's task.*" As such, our definition of adaptive service provisioning can also be interpreted as context-aware service provisioning from the types of changes which must be addressed by an application. However, we also wanted to make a distinction among types of entities which are responsible to adapt the applications with respect to contextual changes (system, end-user, or programmer). As such we call it adaptive provisioning in contrast with tailorable and evolvable provisioning.

– *Dynamic Service Composition*: In dynamic service composition, also known as adaptive or automatic, since we used the concept of SBB, in the execution time, application services must be evaluated and selected for each SBB. As such, in our definition, the service provisioning consists of selecting, configuring and composing application services. However, based on a generic composition framework [123], several service compositions are generated directly from atomic services (i.e., application services) and then evaluated based on runtime situations. The configuration of these atomic services is considered as part of the service composition. Therefore, its definition of service compositions from of atomic services is equivalent to our definition of dynamic service provisioning out of SBBs and their corresponding application services. We should mention that dynamic service compositions is more popular than dynamic service provisioning. However, since we have made an explicit distinction between the tailoring and provisioning, we call our approach as a dynamic service provisioning to emphasize service selection and configuration as well as composition.

We consider dynamic service composition approaches as our related work. We provide a general overview of existing dynamic service composition approaches. However, since it is a broad domain and there are many related work, we zoom into a specific type of dynamic service composition which is a hybrid combinations of processed and rules.

3.2.2 Dynamic Service Composition

Dynamic service composition (also know as adaptive or automatic service composition) approaches have been proposed to deal with

contextual changes in the environment. These approaches require minimal or no manual intervention and generally speaking, can be classified in three categories [123]:

– *Workflow composition*: with respect to our definition of the service plan, in workflow composition, a service plan contains a set of SBBs together with the control and data flow among them. This can be done at two levels: static and dynamic workflow generation. In the static workflow generation, the provisioning platform only discovers and binds the concrete services to the SBBs at runtime without changing the control and data flow. Graph-based composition is the most commonly used method for the static workflow generation [99]. In contrast, in the dynamic workflow generation, both the flows and binding are done by the provisioning platform at runtime.

– *AI-planning*: in Artificial Intelligence planning approaches, a service plan contains a set of SBBs including their precondition and effect in the environment [122]. From the system point of view, each SBB is a software component that has input and output data. Therefore, their precondition and effects are their input and output parameters. From the environment point of view, a SBB is an action which can alter the state of the world after its execution. So the world state which is required before the execution of a SBB is its precondition and the state of the world after the execution is its effect. The provisioning platform, based on these two views of precondition and effect for SBBs, conduct runtime reasoning to generate the executable service plan. There are several AI-planning methods to define the precondition and effects of the SBBs such as rule-based methods. The rule-based method use composability rules to determine whether SBBs are composable or not [102].

– *Hybrid Service Composition*: these approaches use both workflow composition and AI-planning methods. A combination of workflow and rules has been introduced in [34]. By applying this method in our provisioning platform, the service plan is broken down into two parts: (a) The first part shows a basic and simple process model of the SBBs and it includes their data and control flows, and (b) the second part has a set of rules to determine how the basic process model can be updated based on the changes at runtime. These rules includes both composition and configuration information.

In the scope of our research, the hybrid service composition is a combination of processes (as a workflow composition) and rules (as an AI-planning).

3.3 Hybrid Service Composition: Processes and Rules

Although there has been a long discussion to define a balance between processes and rules to model information systems in hybrid service composition, there was no doubt that a combination of processes and rules would be useful to address different types of business goals [171]. Hybrid service composition approaches can be classified in two categories: (a) extending the process logic specification to support rules, and (b) extracting the rules from the process logic and exposing them as an ordinary application service which can be called by the process. We call it extracting (rather than separating) because we assume that the process with embedded rules is given as an input and the rules are extracted and shielded from the process.

For the first category, since standard process logic specification languages (e.g., WS-BPEL) need to be extended, there is a lack of implementation support. One of the earliest works on the hybrid service composition introduced by Charfi and Mezini [34], broke down the service composition problem to several units which can be created, modified or deleted independently. These modules can be implemented by business rules. Aspect-orientation is an alternative approach to implement these rules [33]. This work mainly talks about how to add rules to the process specification, instead of separating the rules from the process and exposing them as a service. Later on, the authors present the design and implementation of AO4BPEL, an aspect-oriented extension to WS-BPEL [35]. However, similar to [34], this approach requires modifying process engines to enable them to handle the aspect-oriented concepts of $pointcuts,\ advices\ and\ aspects$.

In this thesis, we focus on the second category of the hybrid service composition approaches. Some works have been done to study how rules can be extracted from the process logic and exposed as a decision service. In [128] a rule interceptor service has been introduced. This service (1) intercepts all incoming and outgoing web service calls, (2) maps them to business rules, and (3) applies the associated business rules. The proposed business rule broker provides a WSDL interface which can be queried by the BPEL engine. Rosenberg and Dustdar [128] emphasize the importance of integrating rule and process engines. They assume that all the rules must be executed either before or after an interceptor, i.e., activities should be coordinated through

synchronous request-response interactions between the process and rule engines. This approach also assumes that a transformation engine is available that has a predefined data model and XSLT rules to make both process and rule engines understand each other.

In [129], Sapkota and et. al. propose a tuple space to improve the flexibility of their data model. In this approach, data can be added and shared by a process or rule engine on the fly. This approach, similar to [128], also assumes synchronous request-response interactions between the process and rule engines. Therefore, the process can call the decision service only at its certain decision points. Comparing with [128, 129], our approach supports asynchronous notification between the process and decision service in addition to synchronous request-response interactions.

In [46], Döhring and et. al. introduce several rule-based adaptation patterns. These patterns can be applied based on the events which are notified by a rule engine. The authors mainly focus on how the adaptation can tailor the workflow based on contextual changes at runtime. However, the interaction between the process and rule engines is limited to some intermediate events which are predefined in the process. In other words, this approach provides event-based interactions between the process and rule engines rather than providing a decision service.

In [3], Adams and et. al. introduce the adaption patterns as a set of self-contained sub-processes, i.e., worklets which can be selected dynamically at runtime based on contextual circumstances. This work mostly emphasizes how the worklets and their corresponding rules can be specified and associated with each other. Later on, in [4], Adams and et. al. add a new interface to the process engine to support exception handling. However, it does not define the service interface for regular transactions (to execute different types of rules) between the process engine and rules engines. To design and implement a service template for executing rules, the specification of this service interface is required. Moreover, when a new event is added, the corresponding process needs to be changed, for instance, by adding a new event catcher.

3.3.1 Context-aware Rule-based Approaches

The feasibility of Logic-Based Models like rule-based logic, to provide context-aware service composition have been shown in several works [164] [67] [136]. We believe that the context model of a environment like a care home can be modeled by a simple contextual modeling approach like Key-Values model and Markup Scheme Models which are in compliance with rule-based approach [136].

3.3.2 Process and Rule Engines

Regarding the implementation of a hybrid service composition approach, we conducted a survey to see available existing tools. Based on our survey, there is strong industry support for combing process and rules to provide dynamic service composition. We have found several process and rule engines which can be used together to add flexibility to the business processes. The idea of combining process and rules is similar to our idea of the hybrid service composition approach. Table *3-3* shows a number of available process and rule engines. Process engines can be used to execute the processes and rule engines can be used to execute rules.

Table 3-3 Some of the available process and rule engines

Process engine
Drools Flow
Apache ServiceMix (Apache ODE)
WebSphere Lombardi Edition
Cordys BPM
ActiveVOS

Rule engine
WebSphere ILOG
JRules
Drools Expert

We used WebSphere Lombardi Edition [74] to model the orchestration of homecare services. Because IBM is one of the partners of U-Care project and provides the required license and technical support. As we explained before, the application services are provided in the high-level communication protocols and can be directly interact with Lombardi process engine like SOAP-based web services. Another alternative would be Apache Service Mix [17] as a lightweight service bus to support resource-constrained devices installed at the care homes. It supports Apache ODE [16] for the process engine. There are also other options for the process engine such as Drools Flow [85], Cordys Business Process Management (BPM) [38], and ActiveVOS [77].

We used WebSphere ILOG JRules [75] to model our configuration and composition rules. The WebSphere ILOG JRules has been employed to model the rule-based medical protocols such as Arden Syntax for MLM [134]. It enables the care-givers to define the rules in natural language which is suitable for non-technical care-givers. There are other options such as Drools Expert [84] and Jess [52] as a Java rule engine.

Chapter 4

Requirements

"No matter how much you want it to be a technical problem, it's a people problem."
— Gerald Weinberg

To design our dynamic homecare service provisioning (DHSP) platform, we first identified the functional requirements that a homecare service provisioning platform should provide. The homecare applications are designed by an application programmer with respect to the identified functional requirements from care-receivers and care-givers who participated in our field test. Further, we identified non-functional requirements from all stakeholders who are involved in the provisioning of the homecare applications in the field test. The non-functional requirements affected the design of the homecare applications and thus, how the applications satisfied the identified functional requirements. The requirements were identified by interviewing of the stakeholders (mainly functional requirements) and literature study of existing homecare systems (mainly non-functional requirements). Several non-functional requirements have been mentioned in the work of McGee-Lennon [101] and Kleinberger and et. al. [90]. However, with respect to the scope of our research and the identified functional requirements, we only mention some of those non-functional requirements.

This chapter is organized as follows: Section 4.1 explains our methodology to identify both functional and non-functional requirements on the DHSP platform. Section 4.2 classifies the identified functional requirements for several homecare applications and introduces our third-party service providers and application programmer. Section 4.3 explains the non-functional requirements which are related to our research scope and might affect our homecare applications. Section 4.4 explains the functionalities of the homecare

applications with respect to the functional requirements. Section 4.5 discusses which of the existing dynamic service provisioning approaches suits the identified homecare requirements the most.

4.1 Methodology

We used both stakeholder interviews and literature study techniques as our requirement elicitation methodology to identify the requirements on the DHSP platform. The functional requirements were identified by interviewing the stakeholders and with respect to our homecare application scenarios. The non-functional requirements were identified both by the interviews and literature study on other existing homecare systems. Use cases and sequence diagrams were employed to document the identified functional requirements. The non-functional requirements were documented in a textual format as a bullet list. With respect to our research scope (explained in Section 3.1.2), some requirements, for instance requirements on applications interface usability [72] and information interoperability [21, 51, 126], were considered out of our research scope, and thus are not listed in this chapter.

4.1.1 Stakeholder Interviews

To identify functional and non-functional requirements on our DHSP platform, we interviewed the stakeholders who were supposed to use or to provide the application services of the DHSP platform. As explained in Section 2.4, the DHSP platform has five types of stakeholders: (1) care receiver (as the end-user), (2) care giver (as the end-user and domain expert), (3) programmer (who creates the application logic and the tailoring platform), (4) third-party service provider (which provides third-party application services), and (5) infrastructure provider (which provides the software, hardware and communication infrastructure for the DHSP platform).

Due to privacy issues, we could not interview the care-receivers, and instead the care-givers on behalf of the care-receivers conveyed the care-receivers' opinions to us. Moreover, the care center, where our DHSP platform is hosted, also provided the required infrastructures to run the platform. Therefore, the care-givers, as the employee of the care center, also represented the infrastructure provider and explained its requirements in our interview.

To identify the functional requirements, we have performed two interviews with several care-givers in a focus group manner [93]. All the care-givers sat together, and answered and discussed our open ended questions. Since the care-givers used the DHSP platform

through the applications and tailoring platform, which are provided by the programmer, we interviewed the care-givers together with the programmer.

Based on the results of the first interview, the programmer identified the common homecare tasks and their corresponding applications [107]. Then we had several in-depth interviews [89] with the programmer to design the homecare applications and mock-up interfaces which have used by the care-givers. In our second interview with the care-givers, we, together with the programmer, validated our application design and accordingly updated it.

During both interviews with the care-givers, we also asked open-ended questions regarding the non-functional requirements for instance, the acceptable application response time. Then, we listed all these non-functional requirements as natural text and sent them back to the care-givers by email to validate the non-functional requirements as well.

After the validation of functional and non-functional requirements with the care-givers, we had several focus group [93] and in-depth interviews [89] with our third-party service providers and programmer to identify their requirements (both functional and non-functional) as well.

The functional requirements first, classified as several use cases that represents the required homecare applications. Then, we drew a sequence diagram for each use case to show how the application service providers, the tailoring platform, and the DHSP platform should interact to deploy and to run the corresponding homecare application.

4.1.2 Literature Study

We also did a literature study to investigate what non-functional requirements have been identified or addressed by the existing homecare systems. We only listed the non-functional requirements which are related to our research scope and could affect our homecare applications and their functional requirements. The non-functional requirements were documented in a textual format and shown as a bullet list.

4.2 Interviewees

Based on the common homecare tasks [107] that has been identified by the programmer, we classify the functional requirements into three homecare applications which are explained as follows:

– *Vital-sign Monitoring (VsM) Application*: VsM application is an example of a monitoring and reminder homecare system which is explained in Section 3.1.1. The care-receiver wants to measure his vital-signs (e.g., weight) according to a specific scheduled time plan. A corresponding care-giver shall respond to situations in which the care-receiver does not follow his plan or his measured vital-sign is not in a normal range. After the application deployment, the care-receiver can see his schedule on the calendar service running on his Tablet PC. The VsM application starts based on a the scheduled time, and reminds the care-receiver, possibly several times to measure his vital-signs on time. If the care receiver does not measure his vital-sign according to his scheduled time plan on time, or if his vital-signs are not in the normal range, the application sends an alert to the care-giver. To view the vital-sign measurement history, the care-giver uses the vital-sign reporting service on her Tablet PC. We considered three types of VsM applications: blood pressure (BP), oxygen saturation (OX) and weight (WT).

– *Medication Monitoring (MdM) Application*: The MdM application is an example of a monitoring and reminder homecare systems which is explained in Section 3.1.1. The care-receiver should take a medicine at a specific time according to his scheduled time plan. The care-giver should respond to situations in which the care-receiver forgets to take his medicine according to his plan. We have two types of MdM applications: using an automatic medicine dispenser and using a manual medicine dispenser.

– *Social Activity Monitoring (SaM) Application*: The SaM application is an example of social interactive homecare system which is explained in Section 3.1.1. The care-receiver wants to be informed about his favorite social activities for instance, watching movie or drinking coffee in a group. The (social) care-givers should be able to define different social activity groups and to assign the interested care-receivers to one or several of these groups. Then the care-givers can schedule several social activities for each of these groups such as to watch a specific movie every Saturday nights. The SaM application sends a reminder to the interested care-receivers whenever an activity of their groups are active.

In the remaining of this section, we introduce the interviewees whom the requirements were identified from.

4.2.1 Third-party Service Providers

To run the aforementioned three types of homecare applications, we utilized a set of application services. In the U-Care project, we had three third-party service providers that provided the required application services in addition to the alert service as an internal application service of the DHSP platform.

The application services namely calendar, reminder and reporting services were provided by the Biomedical Signals and Systems (BSS) group of the University of Twente [140]. These services were running on the Tablet PCs available to the care-givers and care-receivers.

The vital-sign measurement services namely blood pressure, weight, oxygen saturation services were provided by MobiHealth [106] company. Care-receivers used the vital-sign measurement devices at their care homes, which were connected to a server running at MobiHealth. The MobiHealth server forwarded the vital-sign measurement values to the DHSP platform.

The automatic medicine dispenser was provided by the Innospense [78] company. Care-receivers used the automatic medicine dispenser, which was connected to a server running at Innospense, to take their medicine on time. The Innospense server forwarded the medication intake information (e.g., time stamps and care-receiver IDs) to the DHSP platform. For the manual medicine dispenser, we used a normal physical box in combination with the reminder and alert services.

4.2.2 Tailoring Platform Provider

If the application logic needed to be updated manually to address unforeseen changes, a programmer modified the application logic and accordingly, updated the tailoring platform. The tailoring platform was provided by the Information System (IS) group of the university of Twente [141]. Since the programmer from the IS group was responsible for designing the homecare applications, we interviewed with the care-givers and third-party service providers together with the programmer. The tailoring platform was running as an application on end-users' Tablet PC available only to the care-givers.

4.3 Non-functional Requirements

Several non-functional requirements affect the way that the stakeholders interact with each other and the DHSP platform. These non-functional requirements have been identified during our interviews and literature

study, and are explained as follows:

– *Non-intrusiveness*: In the homecare domain, due to the limited ability of the care-receivers to use ICT-based services, the homecare services should be non-intrusive. This implies that the platform should not often ask care-receivers for interventions, such as whether they prefer a system decision or not. In other type of AI-planing approaches like TLPlan [149], the runtime reasoning is based on the real world model. This provides a highly complete adapting method to the unexpected changes. In this approach, inquiry from the end-users to make sure about the correctness of the runtime decisions, plays an important role. However, based on our interview with the care-givers, we concluded that the care-receivers are not willing (or not able) to answer these system inquiries.

– *On-the-fly Tailorability*: The care-givers wants to change the homecare applications on-the-fly so that the applications update their behaviour without interrupting the running instances of that application. This is important when several instances of an application are running for different care-receivers.

– *Light-weight Evolvability*: The programmer wants to modify an application logic, if it is required, with minimum efforts. Moreover, the re-deployment time of the modified application logic should be as short as possible. The programmer also prefers to define a homecare application in several modules and modifies modules independent of each others for improving evolvability.

– *Learnability*: The care-receivers such as elderly people have difficulties to learn how to use an end-user interface. The care-givers or volunteer people should help them to learn how to use the system and this needs lots of learning efforts. If an application service is replaced by another one, the system should be used in the same way as before to reduce the learning efforts. For instance, switching from manual medicine dispenser to automatic dispenser should not change end-user interfaces of the calender and reminder services.

– *Data Ownership* One of the requirements, which has not been mentioned in the literature to the best of our knowledge, is that the care center wanted to have the ownership of data such as vital-sign records. Therefore, the care center only agreed that the DHSP platform stores the data if it is hosted within the care center and not

the third-party service providers which are located outside the care center. Later on, the care center also agreed that the DHSP platform can be hosted outside the care center if the data will be deleted after the field test. In practice, a care center may not agree with storing the care-receiver's data anywhere outside of the care center.

– *Acceptable Response Time*: The homecare system is a real time interactive system. Thus, the end-user should get response during a system task for instance, successful deployment, receiving an alert or vital-sign records shortly. In our interviews, the care-givers mentioned that the application adaptation must be done *immediately*. For instance, an alert must be delivered *immediately* after the vital-signs measurements in case the values are either higher or lower than a predefined threshold. We tried to quantify that and we found out that less than one minute would be considered as *immediately* by the care-givers.

– *Limited Human Resources*: There are not enough care-givers to support the care-receivers. Thus, the less demanding human resources for any solution in the homecare domain plays an important role in its success. The care-givers asked us explicitly to design the system somehow that it takes their time for creation or deployment or provisioning of the homecare application as less as possible. For instance, although confirming an alert message by a care-giver could improve the reliability of the system, we do not ask them to to that.

– *Limited System Resources*: In homecare systems, only one or two care-receiver(s) are living in each care home. Therefore, it is not feasible to have a dedicated provisioning platform for each care home. The care center prefers to pay only for one provisioning platform and then it is shared by all the applications and care-receivers. It requires a light-weight service provisioning approach and thus a complicated advanced semantic reasoning approach would not be suitable solution. We have observed during the field test that using service on demand, in which the care center can pay for the software and hardware infrastructure based on the number of its care-receiver, is highly desirable in the homecare domain. This requirement still remains as a challenge and should be addressed in our future work.

– *Accountability*: The homecare system is a safety-critical [94] system and every system activity (e.g., sending reminder or alert)

is attributable to an entity (person or service provider) which is responsible using or providing that activity precisely. For instance, when a reminder message is not delivered to a care-receiver and this causes a risk, it must be clear whose responsibility is: (a) the DHSP platform because of not sending the reminder or (b) the service provider which does not deliver the message to the care-receiver although it receives the reminder message or even (c) the care-receiver who receives the reminder message but ignores it.

– *Accurate Adaptivity*: The homecare provisioning platform and the services running on top of it, are classified as safety-critical [94] systems. It means that any system malfunctioning could lead to loss of life. The malfunctioning can arise from the hardware failure as well as wrong decision which are made based on the service plan to adapt the homecare applications at runtime. Therefore, care-givers must be able to accurately control the behaviour of the applications at runtime.

– *Light-weight Integrability*: The third-party application service providers used different system specifications for instance, message format, interaction patterns and implementation platforms. They wanted to integrate with the DHSP platform without changing their system specifications. Beside, they preferred to be independent of other application services and changing one application service in a composite application does not affect their application services.

– *Ethical Laws*: This requirement is imposed by the ethical laws [32] in the homecare domain. Based on this law, the vital signs data such as blood pressure measurement values must be anonymously transfered among different service providers. Hence, during the transportation, none of the services (and their providers) knows to whom the measured data belongs. Only the end-users' application which directly interact with care-giver and care-receiver can decrypt the anonymous data, thus they are aware of the real owner of data. Moreover, third-party service providers such as MobiHealth are not allowed to keep or store the vital signs. Therefore, MobiHealth deleted vital-sign values immediately after forwarding them to the DHSP platform.

– *Medical Protocols* Medical guideline and protocols have been employed by care-givers to provide care services to care-receivers in a standard way [42]. There are several methods to support computer-based modeling medical protocols: (1) Rule-based (2)

workflow, i.e., care-flow (3) task network [63].

Rule-based medical protocols like Arden Syntax for Medical Logic Modules (MLM), which is part of Health Level Seven (HL7), has been employed to facilitate knowledge sharing among care-givers, for instance, for COPD treatment [134]. With respect to our application scenario, MLM can be used for COPD treatment to evaluate medical criteria, and, if appropriate, perform an action such as sending a message to a care-giver [134].

Care-flow as a process defines which task needs to be executed in which order [63]. To implement a careflow complaint system, a workflow management system is considered as a solution [121].

Task network languages are defined based on an ontology of task. Several task network languages like PROforma, are based on a process definition language which make them similar to the care-flow languages method [63]. However, they have their own components ontology such as tasks and decisions.

4.4 Functional Requirements

We classify the functional requirements with respect to the three aforementioned homecare applications. For each application type, first we show a use case to explain the functionalities and the involved stakeholders of the application. The stakeholders could be: (1) care-giver, (2) third-party service provider (BSS, MobiHealth and Innospense), or (3) tailoring platform provider (IS group). We do not show care-receivers, because they only interact with the DHSP platform through the third-party application services and thus, everywhere we have a third-party service provider, we can think of a care-receiver as well. Nevertheless, we show care-givers because they might use the DHSP platform either through the third-party or internal application service, or the tailoring platform.

4.4.1 VsM Application

For the vital-sign monitoring (VsM) application, we have four activities (functionalities) namely deployment, monitoring, updating and view as shown in Figure *4-1*. Since the VsM application is a composite application, to provide each functionality, several stakeholders should interact with each other. These functionalities and their involved

stakeholders are explained as follows:

– *Deployment*: a care-giver through the tailoring platform, provided by IS, creates or tailors a service plan. Then IS group forwards all the information of the created service plans to the DHSP platform and eventually receives an acknowledge that the deployment is done successfully. The IS wants to be independent of the application services and only knows the SBBs and their configuration parameters. The DHSP platform, based on the configured SBBs of the service plan, sets the calendar events on BSS calendar service and as an acknowledgement the DHSP platform receives the created calendar event Ids (identifications). The BSS shows the calendar events to the care-receivers on their Tablet PCs.

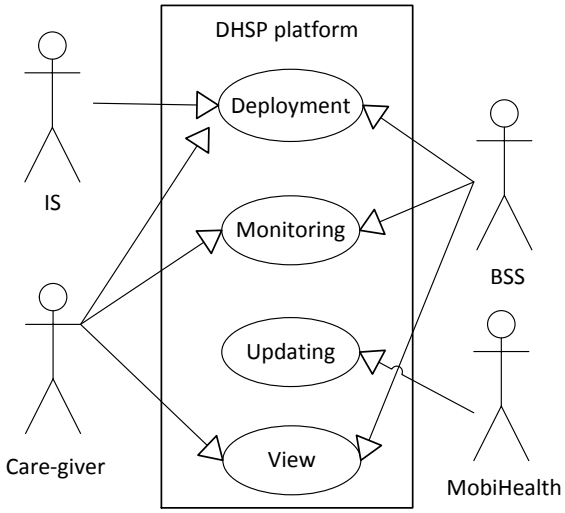

Figure 4-1 : The VsM application functionalities and involved stakeholders.

– *Monitoring*: the BSS calendar notifies the DHSP platform to execute a service plan at its scheduled time. The DHSP checks if it has already received the measured vital-sign values by the care-receivers, and if not, it sends a reminder to the reminder service of BSS. The reminder is context-aware for instance, supporting different modality based on the care-receiver's situations at runtime. The BSS shows the reminder message to the care-receiver. For the *being non-intrusive* non-functional requirement (explained in Section 4.3), the care-receivers do not need to acknowledge that they see the reminder messages. However, BSS should acknowledge that it receives the reminder messages due to the

accountability requirement. If after sending several reminders, no measured vital-sign value is received by the DHSP platform, it will send a context-aware alert to the care-giver indicating that a specific care-receiver forgets to measure his vital-signs. With respect to the *limited human resources* non-functional requirement and thus not assigning extra tasks to the care-givers, they are not obligated to acknowledge seeing alert messages. We should take into account that this might be in conflict with the *accountability* non-functional requirement. However, during the interviews with the care-givers, they indicated that to not to acknowledge the alert messages.

– *Updating*: MobiHealth forwards a measured vital-sign value to the DHSP platform whenever the care-receiver does a measurement. MobiHealth is not allowed to store the vital-sign data due to the *ethical laws* requirement. Beside, MobiHealth prefers to have as less integration efforts as possible (the *integrability* requirement) and since, MobiHealth has implemented an asynchronous notification in another project, uses the same interface to interact with the DHSP platform. As such, we separate updating from monitoring functionality to support asynchronous update of vital-sign measurement. If we did not have this imposed requirement, the updating could be done as part of the monitoring by synchronous request-response query from MobiHealth.

– *View*: a care-giver or a care-receiver can see the measured vital-sign records. For care-receivers, seeing the records encourages them to measure their vital-signs on time. For care-givers, seeing the records helps them to prescribe a precise treatment for the care-receivers and may re-tailor the service plan. The vital-sign records are only stored by the DHSP platform with hashed care-receiver IDs due to the *data ownership* and *ethical laws* non-functional requirements. Therefore, BSS should query the DHSP platform for the the vial-sign records for each request coming from the care-givers or care-receivers.

Figure *4-2* shows how the stakeholders interact with each other and with the DHSP platform according to the four functionalities. The sequence diagram only shows the on-line interactions during the provisioning time. We also have off-line interactions for instance, to share the care-receiver IDs and their hashed values with MobiHealth, IS and BSS. In Section 4.3, we explained that using hashed ID is defined with respect to the *ethical laws* requirement. These hashed values are used for the communication with the DHSP platform.

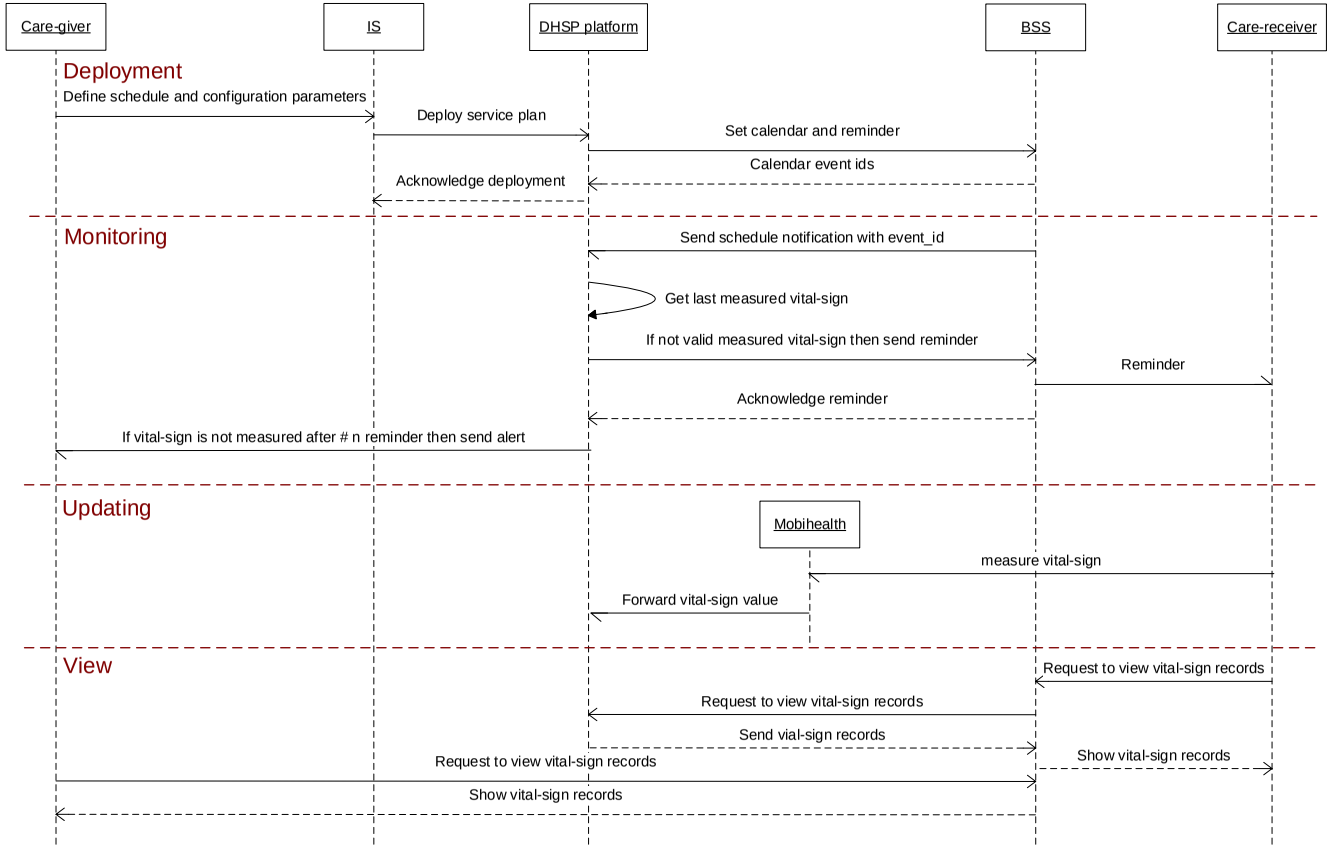

Figure 4-2 : The interaction of the stakeholders to provide the functionalists of the VsM application.

4.4.2 MdM Application (Manual Medicine Dispenser)

For the medication monitoring (MdM) using manual medicine dispenser, we have four functionalities as shows in Figure *4-3*. Similar to the VsM application, to provide each functionality of the MdM application, several stakeholders should interact with each other. Figure *4-4* shows how the stakeholders interact with each other and with the DHSP platform according to the four functionalities. These functionalities and their involved stakeholders are explained as follows:

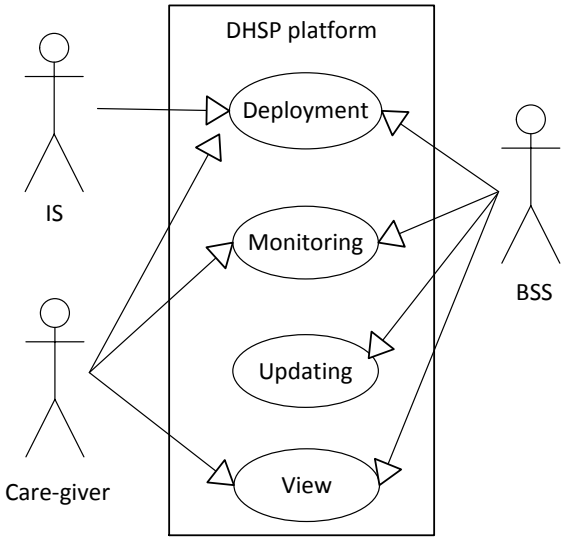

Figure 4-3 : The MdM application functionalities and involved stakeholders (manual medicine dispenser).

– *Deployment*: this functionality is exactly the same as deployment functionality of the VsM application. The care-giver and IS group use the same tailoring platform for all the homecare applications. The care-giver configures the same SBBs (e..g., calendar) used by the VsM application. In addition, the care-giver configures the medicine dispenser SBB.

– *Monitoring*: similar to the monitoring functionality of the VsM application, the DHSP (a) receives notification from the calendar service of BSS to execute an application, (b) if medication is not taken sends a reminder message, and (c) if the medication is not taken after several reminders, sends an alert to the corresponding care-giver.

– *Updating*: for the manual medicine dispenser, we use a simple box. A care-receiver should take one of the medication from this box and press a button on his Tablet PC to indicate that the medication is taken. Then BSS sends an acknowledge indicating that the care-receiver took his medication to the DHSP platform. For the sake of the $accountability$ requirement, the DHSP platform sends back an acknowledge to the BSS indicating that the data is received. Similar to the reasons mentioned for the VsM application, we separate the updating from the monitoring functionality.

– *View*: a care-giver or a care-receiver can see the medication taken records. Similar to vital-sign records, the medication taken history are only stored by the DHSP platform with hash care-receiver IDs due to the $data\ ownership$ and $ethical\ laws$ requirements. Therefore, BSS should query the DHSP platform for the medication records for each request coming from the care-givers or care-receivers.

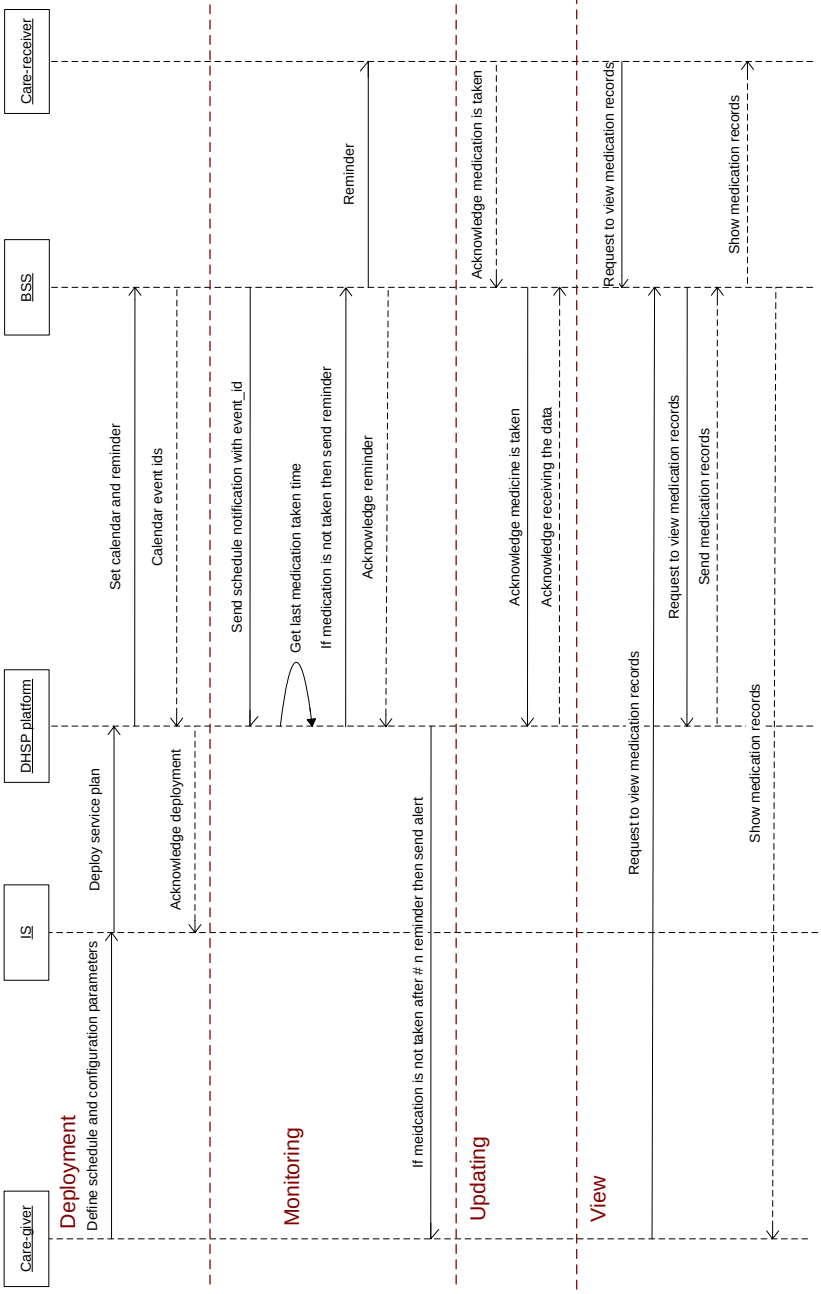

Figure 4-4 : The interaction of the stakeholders to provide the functionalists of the MdM application (manual medicine dispenser).

4.4.3 MdM Application (Automatic Medicine Dispenser)

We also implemented the MdM application using an automatic medicine dispenser to compare its success rate with manual medicine dispenser. The care center was interested to see if using the automatic medicine dispenser is worth paying its extra expenses. For the medication monitoring (MdM) application using automatic medicine dispenser, we also have four functionalities as shown in Figure *4-5*. Compared to previous section, we have one extra service provider, Innospense as the automatic medicine dispenser provider. Due to the $learnability$ requirement, it is difficult for the care-receiver to learn how to use different user interfaces for manual and automatic medicine dispensers. Therefore, although the automatic dispenser has its own schedule interface, reminder and alert, we used the same application services as manual medicine discerner (i.e., calendar, reminder, records history and alert). Figure *4-6* shows how the stakeholders interact with each other and with the DHSP platform according to the four functionalities. These functionalities and their involved stakeholders are explained as follows:

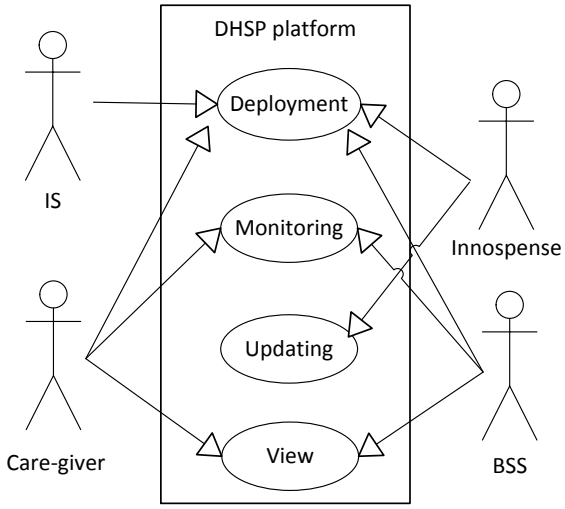

Figure 4-5 : The MdM application functionalities and involved stakeholders (automatic medicine dispenser).

– $Deployment$: from the IS and BSS points of view, it is exactly the same as deployment functionality of the MdM application using manual medicine dispenser. This is one of the benefits of having the DHSP platform to shield the underlying technologies form the service providers and the programmers. To use the application

services of BSS, the DHSP platform deploys the schedule to both BSS and Innospense.

– *Monitoring*: is exactly the same as monitoring activity of the MdM application using manual medicine dispenser.

– *Updating*: Innospense has its own web portal to manage the automatic dispenser. Since the web portal follows the client-server architecture and Innospense does not want to change its service specification due to the *integrability* requirement, we have to implement a request-response method to get the information about latest taken medication. However, we need to call the updating for both view and monitoring functionalities and therefore, we keep it still separated than monitoring functionality.

– *View*: a care-giver or a care-receiver wants to see the medication taken records. Similar to MdM application using manual dispenser, BSS should query the DHSP platform for the medication records for each request coming from the care-givers or care-receivers. Due to the *learnability* requirement, we did not use the reporting page of Innospense web portal and instead used the reporting page of BSS. As such, both care-receivers and care-givers used the same user interfaces for both MdM applications using manual and automatic medicine dispenser.

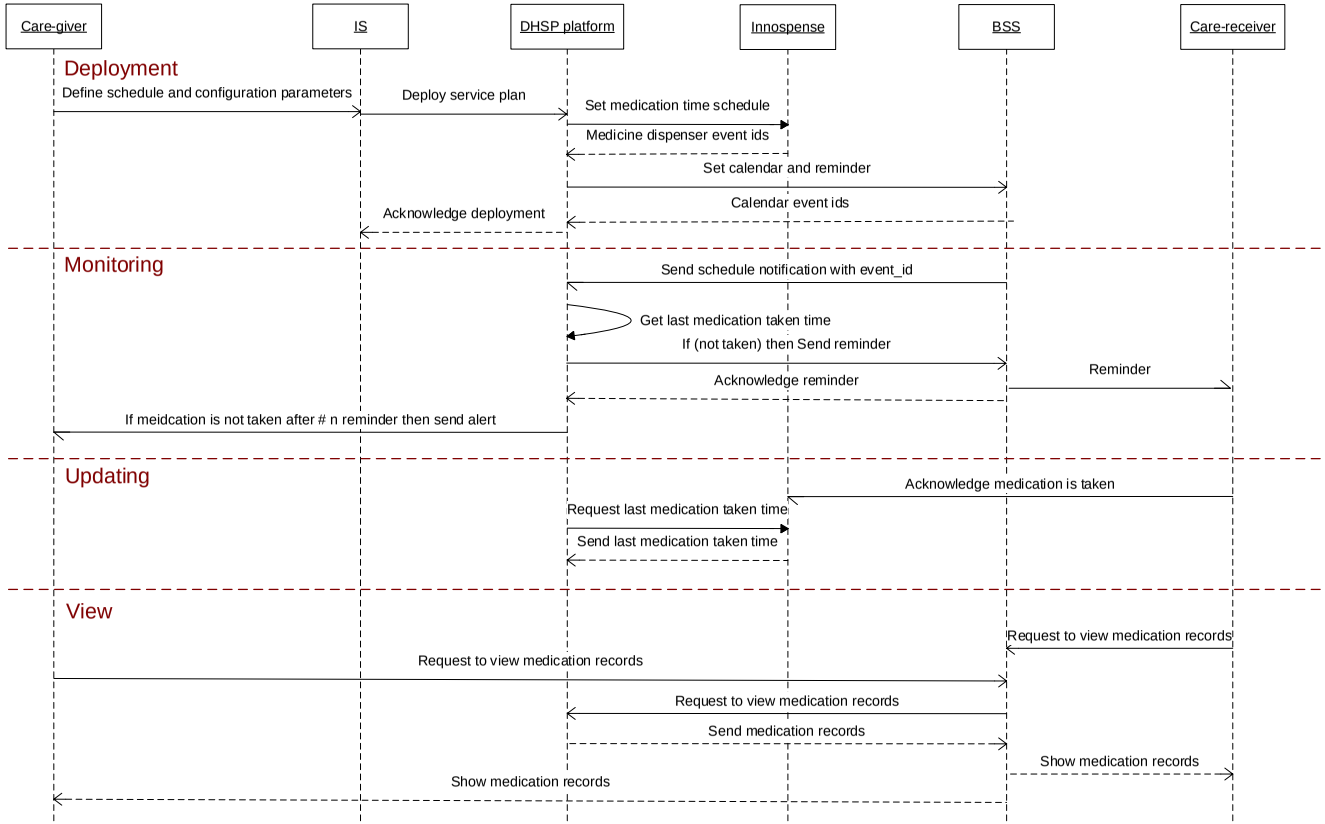

Figure 4-6 : The interaction of the stakeholders to provide the functionalists of the MdM application (automatic medicine dispenser).

4.4.4 SaM Application

For the social activity monitoring (SaM) application, we have three functionalities as shown in Figure *4-7*. The SaM application does not have updating functionality, because there are many social activities and the system became intrusive if it asks the care-receivers to indicate participating in these activities one by one. Figure *4-8* shows how the stakeholders interact with each other and with the DHSP platform accoring to the three functionalities. These functionalities and their involved stakeholders are explained as follows:

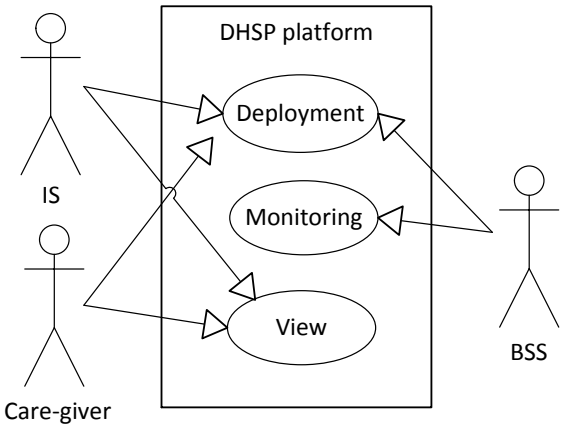

Figure 4-7 : The social activity sequence diagram.

– *Deployment*: a care-giver defines social activity groups, assigns the care-receivers to one or several of these groups and finally schedules several social activities for each group. Then the DHSP platform, sets the social activities events related to a care-receiver in his calendar service. The DHSP platform stores the defined social activity groups and their members.

– *Monitoring*: the BSS calendar notifies the DHSP platform about a scheduled social activity for a care-receiver. The DHSP sends a reminder message to the reminder service of BSS. BSS should acknowledge that it receives the reminder message.

– *View*: a care-receiver can view his social activities directly from BSS. A care-giver can see the defined social activity groups and their members. To do so, the tailoring platform queries the DHSP platform since it stores all the data.

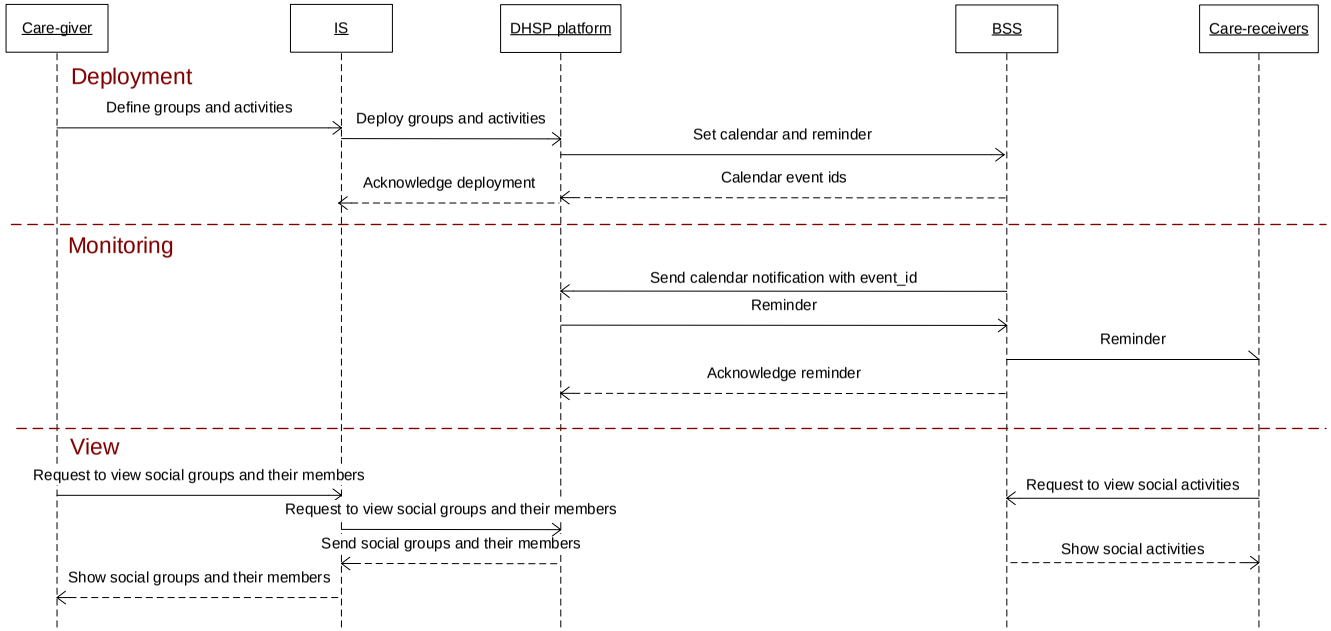

Figure 4-8 : The interaction of the stakeholders to provide the functionalists of the SaM application.

4.5 Discussion

In this chapter, we identify the requirements on a DHSP platform based on interviews with the stakeholders which use the platform, and literature study on existing related work. With respect to these requirements, we conclude that a hybrid service composition using a combination of process and rules is as a promising approach for dynamic service provisioning in the homecare domain. Our conclusion is based on the following observations:

1. There is significant similarity between the hybrid service composition approach and the existing technologies to model computer-based medical protocols.

2. Due to boolean logic of rule-based adaptation, the care-givers can accurately control the behaviour of the system. For instance, Nancy can explicitly determine in the service plan, if Jan leaves the home without attaching oxygen saturation meter, send an alert to the care center.

3. The rule-based approach may compromise the completeness of the runtime adaptation, i.e., not being able to adapt the application behaviour in all the runtime situations. However, the adaptation using rule-based approaches is decisive due to its boolean logic reasoning (either true or false). Thus, this approach does not require the care-receivers to interfere and this suits the non-intrusiveness requirement.

4. There are several industry tools (explained in Section 3.3.2) that can support the hybrid service provisioning. Moreover, IBM as on of the industry partner in the U-Care project provides license and technical support for WebSphere Lombardi process and WebSphere ILOG rule engines.

5. Although other AI-planning approaches such as ontology reasoning, provide better completeness of adaptivity at runtime, they also introduce several challenges [96, 14]. For example, process-intensive reasoning (in contrast with the $limited\ resources$ requirement), data inconsistency due to ontology complexity (in contrast with the $on\text{-}the\text{-}fly\ tailorability$ requirement), and ontology modification and its verification (in contrast with the $accurate\ evolvability$ requirement).

6. Configuration rules can be divided to a set of modules for instance, based on the SBBs and decision points in a process. Furthermore, the possible reconfigurations of a service plan (tailorability) or modifications in the application logic (evolvability), and their effect on the application behaviour are limited to these modules. Therefore, the modularity improve the accuracy of tailorability and evolvability [20], less system errors happen after updating a homecare application by a care-giver or programmer.

7. The modularity of the hybrid service provisioning can decrease the required system resources for re-deploying a service plan. Because, we can only re-deploy only the modified module(s) (e.g., the rules about the reminder SBB) instead of the whole service plan and thus, it can be done one-the-fly according to the updated modules.

The hybrid service provisioning approach is used to select, configure and compose the application services based on runtime contextual situations. For the tailorability and evolvability, the care-giver or programmer can update the process or (a module of) rules or both.

Chapter 5

Dynamic Homecare Service Provisioning Platform

"Innovation distinguishes between a leader and a follower."
— Steve Jobs

In this chapter, we introduce our dynamic homecare service provisioning (DHSP) platform. It is based on a hybrid service provisioning as a combination of processes and rules. The platform hosts several application and infrastructure services to execute the service plans, which are deployed by a care-giver through the tailoring platform. Each service plan consists of several SBBs and application-logic decision making rules (we call them decision rules). Based on the execution of decision rules at runtime, the SBBs are mapped to several application services and then the selected application services are configured and composed. To support this, the platform has several infrastructure services that execute the decision rules and accordingly, select, configure and compose the application services.

This chapter is organized as follows: Section 5.1 explains how a service plan is defined and executed based on the SBBs and decision rules. Moreover, it introduces the SBBs and the types of decision rules which we have used in our DHSP platform. Section 5.2 explains several architectural patterns which are employed by and accordingly, introduces the logical architecture of the DHSP platform. It also explains the deployment process through which a service plan is deployed into the DHSP platform.

5.1 Service Plan: SBBs and Rules

Fig. *5-1* shows how a service plan is defined and executed on the DHSP platform. The service plan, which is deployed through the tailoring platform by a care-giver, consists of two main parts: (a) decision rules and (b) orchestration patterns.

Decision rules are application-logic decision making rules that determine how to select, configure, or compose the application services with respect to the SBBs of a service plan and runtime contextual information. The decision rules of a service plan are executed at runtime based on contextual information for instance, blood pressure and location of a care-receiver.

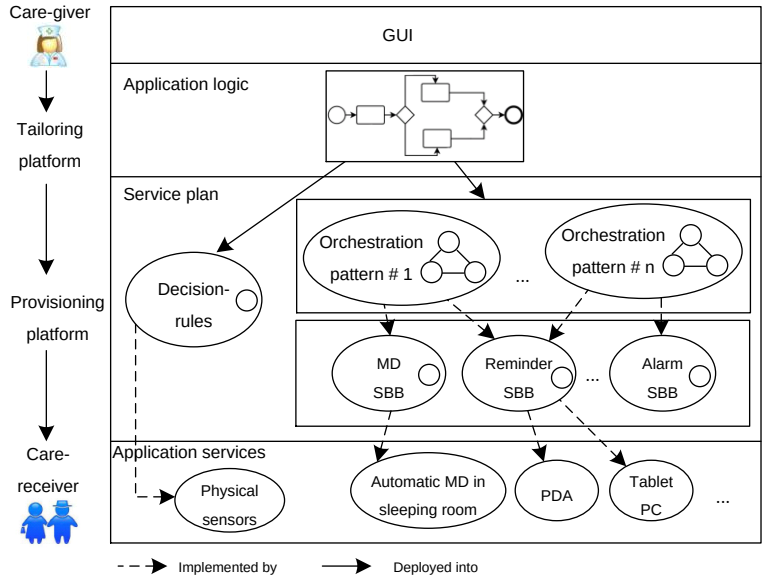

Figure 5-1 : How a service plan is defined and executed on our DHSP platform

Orchestration pattern is part of a service plan and determines how SBBs of a service plan and consequently, the selected application services are composed. Each service plan might have several orchestration patterns (of SBBs), and one of these patterns is selected at runtime based on the decision rules.

To improve the evolvability, we separate the decision rules from the orchestration patterns and expose them as a decision service (explained in Section 6.1). The decision service can be called by or can notify the orchestration patterns. The decision service receives contextual information from physical sensors. These physical sensors are accessible through their corresponding application services. Based

on the data coming from the physical sensors, the output of decision service determines which configurations and orchestration patterns must be selected. For instance, the blood pressure value coming from a blood pressure measurement service is taken as contextual input by the decision service. Then if the blood pressure value is not received at a scheduled time, based on the executed decision rules, the running orchestration of the VsM application might sent a reminder either through the Tablet PC or PDA with respect to the care-receiver's location.

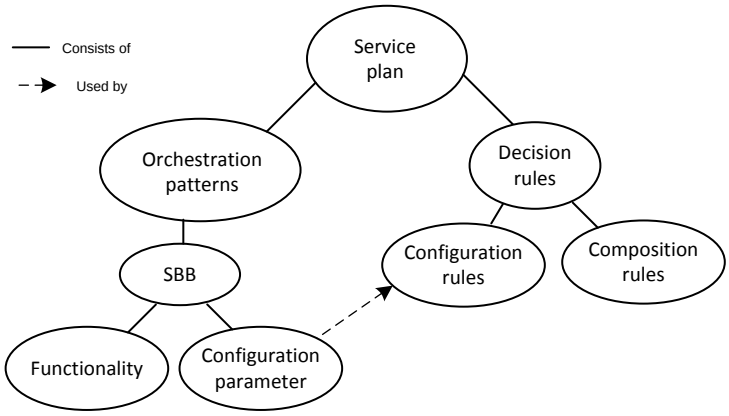

Figure 5-2 : The elements of a service plan

Fig. *5-2* shows the entities of a service plan. The service plan consist of the decision rules and the orchestration patterns. An orchestration pattern consists of one or several SBBs. Each SBB has several functionalities which can be implemented by alternative application services. Moreover, the SBBs have several configuration parameters. The configuration parameters are used by the decision rules for instance, the number of reminder repetition. The configuration parameters of SBBs allow a care-giver to specify different aspects of the SBB such as service operations and user interface modalities. The configuration parameters are used by the decision rules and will be explained in Section 5.1.2.

The decision rules can be either configuration or composition rules. The configuration rules determine how to select or to configure an application service. For instance, a reminder application service on a PDA can be selected for the SBB reminder if the care-receiver is outside the care home or the PDA reminder service can be configured to send the reminder several times when the care-receiver has movement disabilities. The composition rules determine how the selected application services should be composed, i.e., which orchestration

pattern should be selected. In our application scenario, the composition rules are used for tailorability. For instance, for the VsM application, we have two orchestration patterns: one with an activity to call the MdM application and the other one without that activity. If a specific care-receiver needs to take a medication for his high blood pressure, the orchestration pattern with MdM call activity is assigned by the care-giver.

5.1.1 SBBs

With respect to our application scenarios and interviews with both the programmer and third-party service providers (explained in Chapter 4), we have developed a library of SBBs. To define a SBB, we should define its functionalities and configuration parameters. Each functionality of a SBB could support either synchronous request-response or asynchronous notification interactions. Moreover, each functionality uses one or several message formats. The suitable interaction types and message formats are selected based on the requirement of the corresponding service providers. In this section, we explain the interaction types of the functionalities, however, the message formats will be explained later in Chapter 8.

Since a SBB is an abstraction, several alternative implementations may exist that correspond to the same SBB. As such, all the functionalities of a specific SBB may not be supported by all its alternative implementations, i.e., application services. For instance, the audio-enabled reminder is not supported by the reminder application service of the BSS service provider. However, we define the SBBs in generic way and shows all the functionalities which have been mentioned in our interviews with the care-givers to design the possible application scenarios.

We present the identified SBBs, their functionalities and configuration parameters in plain English. We used plain English for the purpose of explanation. Later on, in the implementation phase (Chapter 8), the specification will be defined in the system level. The identified SBBs are explained as follows:

1. Reminder: to notify a care-receiver to do something.

   Functionality:
   (a) Send message() : It is a request-response functionality to send reminder message and in return to acknowledge that the reminder is received by corresponding the application service.

Configuration parameters: *string* Message: message to send to the care-receivers, *time* Timeout 1: how long before the schedule send the first reminder, *time* Timeout 2: waiting time between each reminder repetition, *integer* Repetition: Number of reminder message repetition, *list* Modality: in which device or modality to show the message.
Some possible values for configuration parameters: Modality: audio, video, text and vibration.

2. Alert: to inform a care-giver if there is a hazard situation.

   (a) Send alert() : It is a request-response functionality to send alert message and in return to acknowledge that the reminder is received by the corresponding application service.

   Configuration parameters: *string* Message 1: message to send to the care-givers about ignoring reminder messages, *string* Message 2: message to send to the care-givers about high or low vital sign values, *list* Interface: in which interface to show the alert, *person* care-giver(s): to whom send the alert.
   Some possible values for configuration parameters: Interface: phone call, SMS, Google talk and email.

3. Calendar: to manipulate the calendar (i.e., agenda) events of the care-receiver.

   (a) Set agenda() : It is a request-response functionality to set or to edit the schedule of a care-receiver, so he can see the calendar events (schedules) on his Tablet PC, and in return to send back the unique event ID (identification) of each calendar event which has been set by the calendar service.

   (b) Notify calendar event() : It is an asynchronous notification functionality to notify the DHSP platform with an event ID to indicate that the time of one specific calendar event is arrived and then the platform can execute the corresponding service plan for that specific calendar event.

   Configuration parameters: *date* From: from which date to schedule, *date* To: until which date to schedule, *time* Time: for which time to schedule, *list* Repeat: how often this schedule will be occurred (repeated), *list* Location: where the event location is (this parameter is needed for some tasks, e.g., to notify a care-receiver

about a social activity in a specific place). *Some possible values for configuration parameters*: Repeat: every day, once a week, once a month or twice a day.

4. Medicine dispenser: to provides access to the medicine dispenser used by the care-receiver.

   (a) Set schedule() : It is a request-response functionality to set the medication schedule on the dispenser of the care-receiver. So the medicine dispenser device knows when it should dispense a specific medicine and in return, the medicine dispenser device sends back the event ID of the medication schedule.

   (b) Get medication intake() : It is a request-response functionality to get the medication intake timestamps. And in return, the platform sends an acknowledge that it receives the data.

   (c) Notify medication intake() : It is an asynchronous notification functionality to notify the latest medication intake timestamp to the DHSP platform.

   Configuration parameters: *list* Modality: the type of interaction between care-receiver and dispenser to take the medicine from it. *Some possible values for configuration parameters*: Modality: manual dispenser, automatic dispenser.

5. Reporting service: to show the measured vital-sign values or taken medications records to the care-giver or care-receiver.

   (a) Notify record() : It is an asynchronous notification functionality to send the latest measured vital-sign value or taken medication timestamp to the corresponding application service.

   (b) Get records() : It is a request-response functionality to send start and end dates of a time period beside a care-receiver ID and in return, to send all the measured vital-sign values or medication intake timestamps for that time period.

   Configuration parameters: *integer* Records number: the number of records should be shown to the end-user, *date* Archive date: the

records before the archive date are not shown to the end-user. *Some possible values for configuration parameters*: Records number: 10 record per query.

6. Blood pressure measurement service: to provide the latest measured blood pressure and its timestamp.

   (a) Notify blood pressure() : It is an asynchronous notification functionality to enable the blood pressure measurement device to send the measured value and its timestamp immediately after the measurement.

   Configuration parameters: *integer* Diastolic 1, Diastolic 2: to set the threshold for diastolic level, *integer* Systolic 1, Systolic 2: to set the threshold for the systolic level, *time* Validity: validity of last measured blood pressure.
   Some possible values for configuration parameters: Diastolic 1: 55, Diastolic 2: 100, Systolic 1: 80, Systolic 2: 200.

7. Weight measurement service: to provide the latest measured weight and its timestamp.

   (a) Notify weight() : It is an asynchronous notification functionality to enable the weight scale device to send the measured value and its time stamp immediately after the measurement.

   Configuration parameters: *integer* Compare point: comparing the measured weight with this value, *integer* Weight maximum or minimum: the difference of the measured weight with the compare point is higher or lower than this value, an alert message should be send to the care-givers, *time* Validity: validity of last measured weight value.

8. Oxygen saturation measurement service: to provide the latest measured oxygen saturation and its timestamp.

   (a) Notify oxygen saturation() : It is an asynchronous notification functionality to enable the oxygen saturation measurement device to send the measured value and its timestamp immediately after the measurement.

   Configuration parameters: *integer* Threshold 1: under this

value, an alert message should be send to the care-givers, $integer$ Threshold 2: above this value, an alert message should be send to the care-givers, $time$ Validity: validity of last measured oxygen saturation level.
$Some\ possible\ values\ for\ configuration\ parameters$: Threshold 1: 92 and Threshold 2: 100.

5.1.2 Decision rules

Decision rules are used to specify how the homecare application should behave during the provisioning. The decision rules can be either configuration or composition rules. The configuration rules are used to select or to configure the application services. The composition rules are used to select an appropriate orchestration pattern. To conclude, we identify four types of decision rules in the homecare domain which are as follow:

1. $Trigger\ Rules$: As part of the configuration rules, this type of rules are needed to specify when a process (i.e., service plan) should be started. We model this type of rules as Event Condition Action (ECA) rules of the from on event if predefined condition is true do start the process.
   We identify three type of events which can trigger a service plan process:

   - $Time\ event$ based on predefined schedule
   - $Context\ event$ based on the change in the context of a care-receiver
   - $Chain\ event$ based on calling from other processes (the condition part of this event is always true)

   Table *5-1* illustrates three examples of trigger rules.

Table 5-1 : Examples of trigger rules

Rules	Event	Condition	Action	App.
R1	Calendar.VsM	BP.notTaken	After t min start the VsM application	VsM
R2	New BP measurement	BP_SyS > 200	start the VsM application to send alert	VsM
R3	VsM.Called	Always true	start the MdM application	MdM

2. *Mapping Rules*: As part of the configuration rules, this type of rules is used for mapping each SBB to available application services. For instance, the following rules determine which application service should be selected for the reminder SBB.

If Care-receiver.location is outside home *Then* Reminder.device= PDA *Otherwise* Reminder.device= Tablet PC

3. *Configuration Rules*: As part of the configuration rules, this type of rules might specify either (1) the behaviour of a selected application service or (2) the control flow of a selected orchestration pattern. For each selected application service, alternative configurations are used to specify the behaviour of a specific application service. Moreover, in each orchestration patterns, decision points are used to specify the control flow of the process to decide what to do based on the run-time data. The following rules are examples of configuration rules to specify application service behaviour and data flow of an orchestration respectively.

If Reminder.repetition $> n$ *Then* Alarm.send

If Care-receiver.location is outside home *Then* Reminder.timeout 1 = 20 minutes *Otherwise* Reminder.timeout 1 = 5 minutes

4. *Composition Rules*: This type of rules defines which orchestration patterns should be selected for a specific service plan. This can be defined by runtime contextual information

(adaptivity) or by care-receiver (tailorability). In our field test, we only use the composition rules for tailorability. For instance, we have two orchestration patterns for the VsM application, one with MdM call activity (orchestration ID=101) and one without MdM call activity (orchestration ID=102). Then based on the tailoring information (for instance, specify the range of blood pressure to take medication), the correct orchestration pattern is selected. The following rule is shown as an example:

If blood_pressure_range_ to_take_medication.IsSet $Then$
VsM.orchestration_id = 101

As explained, the decision rules are defined using several configuration parameters which are assigned to the SBBs and build a data model. The *data model* is a data schema that defines the structure of the configuration parameters which are used by the decision rules. An orchestration pattern also uses (part of) the data model based on the SBBs that consists of.

Fig. *5-3* shows how decision rules and orchestration patterns use the data model. The data model is instantiated with the values of its parameters at runtime. The decision rules and orchestration patterns of a service plan should be able to read or to write the same data model instance. How the decision rules and orchestration patterns manipulate the data model values is explained in Section 6.1.

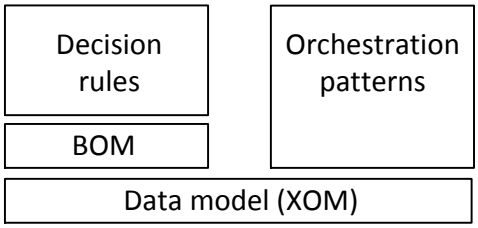

Figure 5-3 : How a data model shared by the decision rules and orchestration patterns

The data model, also known as XOM (Executable object model), can be used by its corresponding BOM (Business object model) and rules templates in a rule editor to create or to tailor a service plan. For instance, a care-receiver in a BOM can be translated to a Java class including several integer, string and boolean parameters in a XOM. The care-giver only sees the BOM and later on the tailoring platform translates the XOM model and deploys it to the DHSP platform.

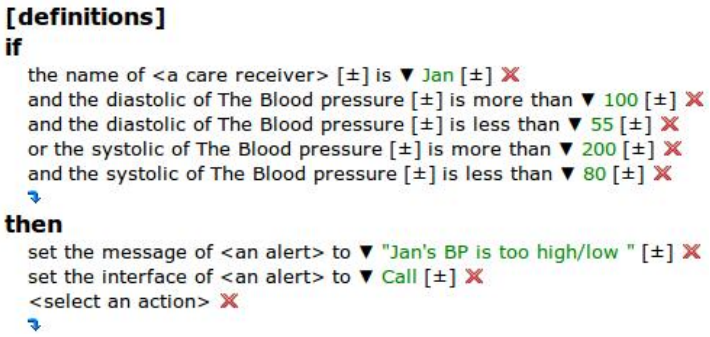

Figure 5-4 : The BOM and rule templates in ILOG rule editor

Fig. *5-4* shows how a care-giver can edit the VsM application decision rules in ILOG [75] rule editor. For instance, the diastolic and blood pressure are the business objects and "*the diastolic of The Blood pressure is more than*" is a rule template. By clicking on the plus or minus symbols, the care-giver can change the objects or their values.

5.1.3 Example Service Plan

With respect to our application scenarios, we illustrate a service plan for a VsM application that monitors the blood pressure of the care-receiver. The service plan of the VsM application should be created and tailored by Nancy (a care-giver) for Jan (a care-receiver) to help him to measure his blood pressure on time. The application starts based on a predefined calendar event and reminds Jan, possibly several times, to measure his blood pressure. If he does not measure or his blood pressure is not in the normal range, the application sends an alarm to Nancy. If his blood pressure is still in the range but the systolic level is higher than 140, the VsM application calls MdM application to remind him to take his medicine.

Fig. *5-5* shows the service plan of the VsM application (for blood pressure monitoring), which consists of an orchestration of SBBs as well as decision rules to specify the behaviour of the application at runtime. We have used this BPMN-like process to define a service plan together with the programmer as a IT-expert stakeholder. Later on, the programmer translates this model to a more understandable tailoring interfaces for the care-givers [168].

The Fig. *5-5* shows an orchestration pattern and its corresponding decision rules. For instance, rule r_0 defines when the application starts, rule r_4 determines how many times to send the reminder and rule r_6 determines to which application service the reminder SBB should be mapped, based on Jan's location at runtime.

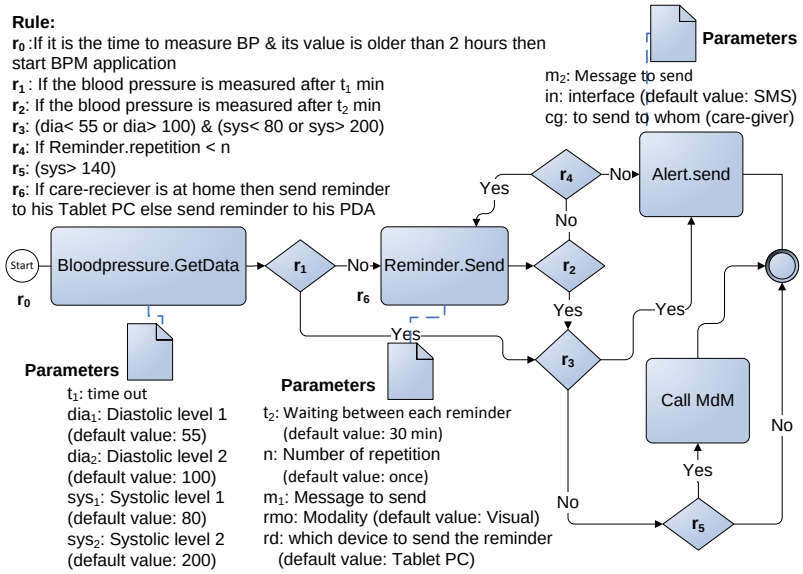

Figure 5-5 : The service plan of the VsM application from the programmer's point of view

5.1.4 Design validation

The service plan, its SBBs and decision rules should be validated by the third-party service providers and the programmer as the interface of the DHSP platform to the outside world. The third application service providers interact with the DHSP platform through the SBBs. In addition to the SBBs, the decision rules and their data model are used by the tailoring platform to deploy a service plan. Therefore, we need to validate the SBBs including their functionalities and configuration parameters with our third-part service providers and programmer before implementation.

After identifying the requirements of the care-givers, first, we, together with the programmer, designed a library of SBBs and their configuration parameters. Then, we had several in-depth interviews with the application providers to identify the functionalities which they can provide with respect to the identified SBBs, their interaction types and message formats.

Accordingly, we implemented the identified SBBs as a set of mock up web service interfaces and asked the service providers to check if they could interact with them. Based on their requirements, we changed some of the message formats and interaction types. For instance, for MobiHealth, we first developed synchronous request-response method to get the measured vital-signs. Later on, we changed to asynchronous call back notification methods using the WSDL which the MobiHealth

provided to us.

Based on the identified SBBs, the programmer developed several mock up tailoring interfaces for the care-givers [168]. Then, we had several in-depth interviews with the programmer to update the identified decision rules and their data model with respect to the changes asked by the care-givers. Then we asked the programmer to create or to tailor test applications. We also asked the service providers to send or receive random data for a few days. At the final stage, we had one session group interview with the programmer and all the service providers to see if the system is ready.

5.2 The DHSP Platform Architecture

With respect to the requirements that exist in the homecare domain, we choose a hybrid service composition, a combination of process and rules, to design the provisioning platform. Therefore, we need a *rule engine* to execute the decision rules based on the contextual changes which are monitored and reported by a component (we call it *context manager*). Accordingly, we need a *process engine* to run the processes of our service plans and coordinate the selected application services. As such our platform the DHSP platform should have three infrastructure components: *process engine*, *rule engine* and *context manger*.

To present our provisioning architecture, first we explain the architectural patterns which are employed by our provisioning platform. Then we describe how the infrastructure services interact with each other and the other platform services to deploy and execute the applications.

5.2.1 Architectural Patterns

Based on our definition, the provisioning platform should address the adaptivity, tailorability and evolvability of homecare service provisioning. For these properties, there are several architectural and design patterns which allow us to reuse of solutions proposed by experienced practitioners for the common problems [2]. The patterns which are employed by our proposed logical architecture are explained as follows:

1. *Adapter*: It is a pattern to enable heterogeneous software components such the application services interact with each other by providing compatible interfaces. Using adapters can address the *integrability* requirement, since it allows the service providers not to change their current implementations. Moreover, the

platform can be used as a service layer for the programmer and service providers to shield the underlying technologies and implementations. This enables the DHSP platform to replace an application service by another one without affecting the application level such as the tailoring platform (the *learnability* requirement). Based on our interviews with the service providers, the application services are accessible through SOAP [150] protocols. In addition, we used SOAP to interact with the WebSphere Lombardi process engine [74] (explained in Section 3.3.2). Therefore, our adapters are only used to provide uniform functionalities with respect to the identified SBBs in the application level and not in the communication level. For instance, we have an adapter to receive a blood pressure (BP) measurement value in any message format, converts it to a unique message for the BP measurement SBB, and then notifies it to the DHSP platform.
In case of internal application services, since they are implemented by the platform provider, there is no need for functionality adaption. For instance, the alert is implemented as an application service to send an alert message to the care-giver. For sake of simplicity, in this thesis, we do not make distinction between application services and adapters. Therefore, an application service could either provide an application service itself or adapt a third-party application service.

2. *Event-Control-Action*: Due to our definition of the adaptivity, contextual changes should be addressed by the application running on the DHSP platform. We chose Event-Control-Action pattern [45] for our DHSP platform to provide the adaptivity of the applications. By using this pattern, we decouple context concern from reaction by means of Event-Control-Action rules.
The application services, which provides contextual information such as the BP measurement service, have publish-subscribe interfaces. For each contextual event, one or several of these application services have been subscribed by the context manager. The context manager notifies the *rule engine* if any contextual event happens. The rule engine can also query the context manager about the current contextual conditions.

3. *Process vs. Rule Engine*: Since we chose the hybrid service composition approach, the provisioning platform employs the rule engine pattern to manipulate the rules. These rules are fired based on the contextual events (triggered by the context manager) or non-contextual events (triggered directly by application services). For

instance, the new blood pressure measurement event is triggered by the context manager as a contextual event and the calendar event is triggered by the calendar service as a non-contextual event.
In contrast with the rule engine, the platform employs the process engine to manage the orchestration of the services, which is more static compared to the contextual decision rules. The rule engine can either trigger a process (orchestration) in the process engine or be queried by the process engine at the decision points of the running processes. The rule engine has several components such as rule repository to maintain all the rules for execution and pattern matcher to decide which rule should be fired based on the event and contextual conditions. In this thesis, since we emphasize on the interaction between the rule engine and the process engine, the internal components of the rule engine are consider out of our research scope.

5.2.2 Overall Logical Architecture

As shown in Fig. *5-6*, the DHSP platform has three subsystems which can be hosted on different physical servers. These subsystems are explained as follows:

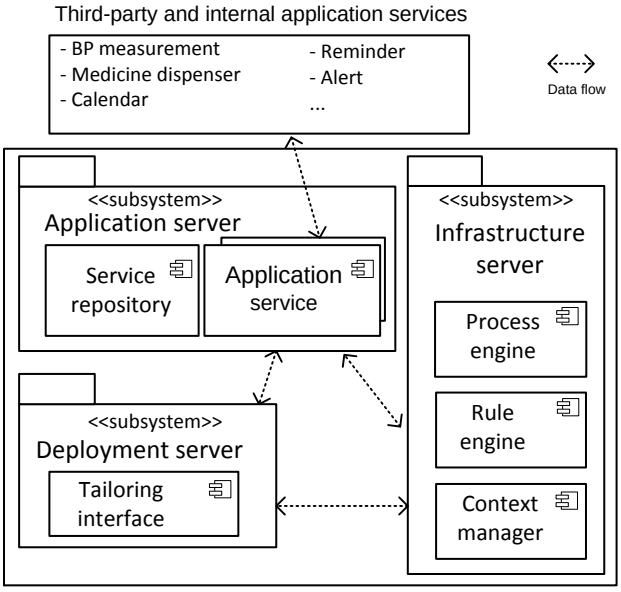

Figure 5-6 : The proposed logical architecture of the provisioning platform

– *Deployment Server*: It hosts a tailoring interface component. The tailoring interface interacts with the tailoring platform. It receives a service plan and sends back an acknowledge to the tailoring platform indicating that the service plan has been deployed. To deploy a service plan, the deployment component (1) instantiate the decision rules of the service plan on the rule engine with the given values configuration of parameters and (2) creates the required context events and their trigger rules on the context manager.
Before the tailoring platform can deploy a service plan, the application logic of that service plan such as decision rule template, the XOM and its orchestration patterns should be created manually by the programmer.

– *Application Server*: It hosts several application services that provide uniform application services out of either third-party or internal application services. It has a service repository to maintain the binding ports and WSDL interfaces of the application services. The application server can be located inside a care home, also known as home gateway, or at back office like a care center. This design choice depends on whether the devices at home are able to communicate with the *application server* at back office using SOAP protocol. In Chapter 8, we explain why our home gateway is located in the back office.

– *Infrastructure Server*: It hosts the three infrastructure components: *process engine*, *rule engine* and *context manger*. These components have several inner and outer interfaces. The inner interfaces can be used by one of the other infrastructure components. The outer interfaces can be used by the application services, service repository and deployment component.

Fig. *5-7* shows all the inner and outer interfaces of the three infrastructure components, The rule engine and context manger have deployment interfaces to deploy a rule and contextual event, respectively (R8,C2). Since we assume all the possible orchestration patterns of a service plan have been created on the process engine before the service plan deployment, the process engine does not have any deployment interface.

The rule engine sends a notification to the process engine either to start an orchestration of a service plan or to inform a running orchestration about new event (R1-P1). In addition, the process engine can call a reasoning service, provided by executing a rule set on top of the rule engine (P2-R2).

Since we used event-condition-action pattern to decouple contextual concern from the actions done by the services, there is no direct communication between context manager and process engine. The process engine can directly call an application service in order to coordinate the orchestration for instance, call a reminder service to send reminder to a care-receiver (P3).

Based on the events defined in a service plan, the rule engine or the context manager subscribes to a set of application services (R7,C3). The contextual events are defined on the context manager during the deployment process. For each of the contextual event, the context manager subscribes to one or several application services. Therefore, the context changes are sent to the context manager through its interface (C1). If a contextual event happens, the context manager triggers the event through the rule engine interface (C4-R5). In addition, if the rule engine needs more contextual information, it can query the context manager (R4-C5). The non-contextual events, like calendar events, are directly defined in the rule engine and the corresponding application service, like calendar service, can notify the rule engine through its interface (R3). During the execution, the rule engine contacts the service repository to know what are the available application services for a specific SBB (R6) to be bound.

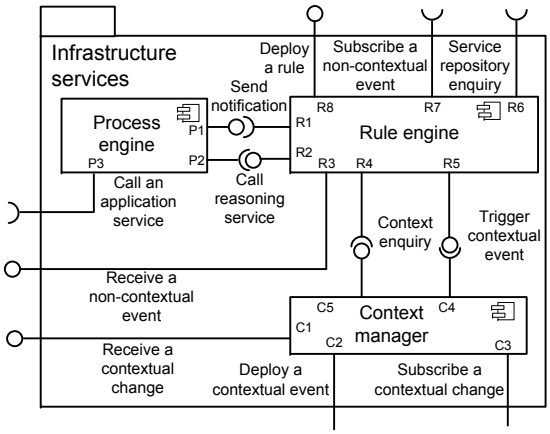

Figure 5-7 : The infrastructure components inner and outer interfaces

The three infrastructure components provide three infrastructure services namely *orchestration service*, *decision service* and *context service* which will be explained in more details in Chapter 6.

5.2.3 Steps Before Running a Homecare Application

Before running a homecare application, it should be created and then deployed to the DHSP platform. In our design, the application logic of a homecare application consists of its *Data Model*, *Decision Rule templates* and *Orchestration patterns*. Therefore, the application logic is created by the programmer through the following steps:

1. The programmer defines the data model (the XOM) on the infrastructure server to be accessible by the context manger, process and rule engines. It defines the structure of the configuration parameters and their possible values.

2. Using the data model, the programmer should define the rule templates on the rule engine. These templates can be instantiated later on during the service plan deployment by the tailoring platform.

3. The programmer creates the orchestration patterns using the SBBs defined on the DHSP platform. In our approach, the orchestration patterns are predefined for each homecare application. Based on our interviews with the care-givers, each application has several alternative orchestration patterns with unique id. So during the tailoring, based on the decision rules and the values of their configuration parameters, an orchestration pattern will be selected for a care-receiver using its id.

After the application logic is created, the care-giver can deploy a service plan using the tailoring platform. During the service plan deployment, the following tasks will be accomplished by the provisioning and tailoring platforms:

1. The decision rules will be instantiated on the rule engine during the service plan deployment. This can be done through the tailoring platform as far as their input and output parameters remain unchanged.

2. Based on the events and conditions of the decision rules of a deployed service plan, the rule engine or the context manager subscribes to several application services. For instance, the service plan of Jan's VsM application has a calendar event, so the rule engine directly subscribes to the calendar service with Jan's ID. The VsM application also needs to know whether Jan's blood pressure is measured or not. So the rule engine also subscribes

to the BP measurement service with Jan's ID. The Subscription is done automatically during the service plan deployment process. Then the deployed application is executed based on its trigger event such as a calendar event.

5.2.4 Why Orchestration Patterns?

We first wanted to manipulate the orchestration, i.e., adding or removing activities at runtime to support the *adaptivity* of the homecare applications. Several hybrid (process and rules) dynamic service provisioning have been introduced that can adapt the orchestration at runtime based on the contextual changes. Generally speaking, they can be classified into two categories: aspect-oriented [35] and worklet-based [3, 46].

Inspired by the aspect oriented approaches, we defined task-depended decision rules (advices) that enables the DHSP platform to manipulate the orchestrations through the decision service [48]. We have defined joint points, i.e., some specific points in an orchestration after or before one activity. Then the joint points are linked to several advices. An advice consists of one or several BPEL [114] activities and their corresponding decision rules. For instance, we have an advice to open the medicine dispenser door (adding one activity), before sending reminder (joint point), if the care-receiver can not use his hands (decision rule).

An aspect is different from a decision rule because beside containing a rule, an aspect also specifies where the rule is applied by its joint points list. Thus, one can separate decision rules from specific processes and can increase the reusability of decision rules.

For the implementation, we introduced an aspect manager implemented by Java script before and after each activity of an orchestration. The advices are stored in separate XML files. However, later on, we have decided to use orchestration patterns instead of manipulating the orchestration at rune time due to several reasons which are explained as follows:

1. *Difficult Tailoring*: We, together with the programmer, found it too complicated for the care-givers to define task-dependent rules (advices). The programmer had difficulty to model these task dependent rules somehow that the care-giver can learn how to configure them. Keep in mind, to address *Learnability* and *Limited human resources* requirements (explained in Section 4.3), the programmer should build a tailoring platform which does not take long time from the care-givers to learn and

to create or to deploy a service plan.

2. *Lack of Industry Support*: To the best of our knowledge, we have not found a standard industrial support to apply aspects on running orchestrations. Therefore, we used the Java script feature of WebSphere Lombardi process to execute the aspect. Java script execution takes longer time than executing the orchestration itself and increased the response time of the homecare application. In addition, using higher computation power contradicts the *limited human resources* requirement (explained in Section 4.3).

3. *Difficulty of Service Plan Validation*: Even with a few predefined orchestration patterns, we, together with the programmer, spent a lot of times to test if all the possible variations of a service pan works without system error. Adding aspects significantly increases these possible variations and we had no tools that can automatically validate them. Not validating all the possible variation of a service plan execution could be very crucial due to the *accurate adaptivity* requirement (explained in Section 4.3).

4. *Tailorability vs. Adaptivity*: In our application scenario, we only needed to change the orchestration for the tailorability and not for the adaptivity. As such, possible orchestrations were defined based on the care-givers' need with respect to exiting *Medical protocols* and was selected at the deployment time.

Our proposed approach can be extended by using worklets as a set of self-contained sub-processes. To do so, the output of decision rules should be defined based on worklets which can be added to an orchestration. We discuss this extension in our future work in Chapter 10

Chapter 6

Decision as a Service

"Good design is obvious. Great design is transparent."
— Joe Sparano

In the previous chapter, we introduced the DHSP platform and its three infrastructure components. We explained how these three components interact with the rest of the platform to deploy and to run a service plan. In this chapter, we zoom into the three infrastructure components and explain how they interact with each other in more details. The three components: process engine, rule engine and context manager, realize three infrastructure services namely *orchestration service*, *decision service* and *context service*, respectively. We separate decision-making rules (decision rules) from application process logic and then expose these rules as a *decision service*, which can be employed by the orchestration service to make adaptation decisions with respect to runtime contextual changes. The interaction between orchestration and decision services are generally performed in synchronous request-response manner [128, 129]. We argue that such an interaction is not efficient to support different types of adaptation at runtime and therefore asynchronous interaction should also be supported. We also explain how the data model can be shared between the infrastructure services by exposing them as a *data model service*.

This chapter is organized as follows: Section 6.1 introduces the architecture to support the decision service and explains how the decision service manipulates the data model values and interacts with the other services. Section 6.2 introduces the decision service template including its service interfaces, their input and output messages, and their message exchange patterns to define the behaviour of the decision service.

6.1 Infrastructure Services

The architecture of the infrastructure server (explained in Section 5.2.2) is shown in Fig. *6-1*. It has four infrastructure services: *orchestration service (OS)*, *decision service (DS)*, *context service (CS)* and *data model service (DMS)*. These infrastructure services are application scenario independent and can be used in any application domain that requires composition of services dynamically. In addition, the figure also shows the service interfaces and the interaction between them.

In this chapter, we only show the four infrastructure services and not the application services. We will show the application services, when we explain the implementation of the architecture later in Chapter 8. We will show the usability of the proposed architecture in supporting the interaction between infrastructure and application services in Chapter 9.

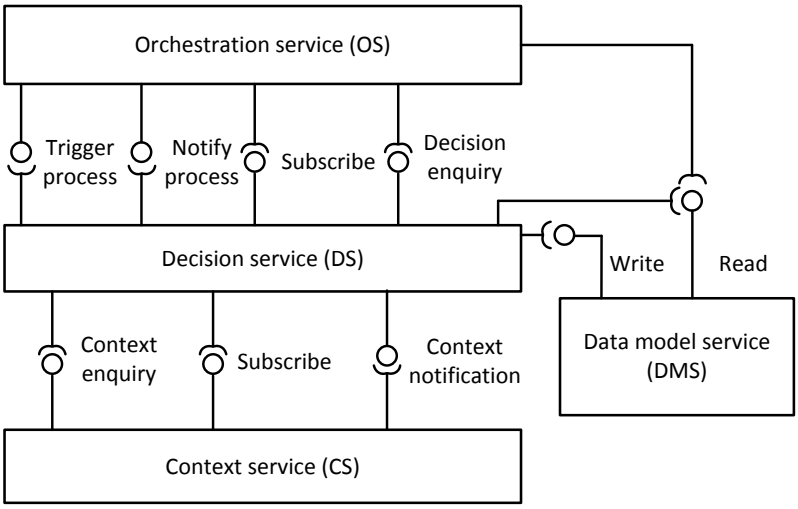

Figure 6-1 : The architecture of the infrastructure server (Fig. 5-6) to support the DS

To clarify our explanation, we repeat the service plan example here in Fig. *6-2*. In the following subsections, we describe the role and responsibilities of each infrastructure service depicted in Fig. *6-1*.

6.1.1 Context Service

The context service (CS) is responsible for aggregating the contextual information from physical sensors which are accessible through a set of application services. As mentioned in Section 2.1, context information is used to characterize the situation of an entity (e.g., person) which is relevant to an user, application or their interaction [43, 44]. For instance, in the homecare domain, location or blood pressure value of a care-

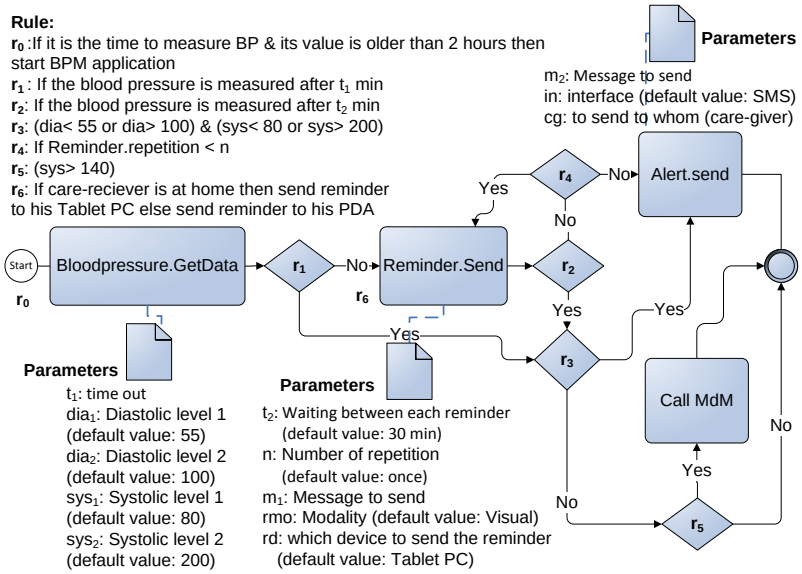

Figure 6-2 : The service plan of VsM application from the programmer's point of view

receiver is the context which is relevant to the VsM application, since this information characterizes (the situation of) the care-receiver. The context service (1) triggers a subscribed contextual event to the decision service (e.g., the event 'Jan leaves home' is subscribed to decide when to use which device to deliver a reminder message to Jan) through the *context notification* interface or (2) responds to the context enquiry coming from the decision service (e.g., to get the latest blood pressure measurement needed for its validation) through the *context enquiry* interface. In order to receive a specific contextual event, the DS should first subscribe for that contextual event through the *subscribe* interface.

We used a simple key-value ontology model [136] to define contextual events and the relationship between them. There are some other approaches such as Markup Scheme Models (for hierarchical data structure), Graphical Models (for graphical representation feature) and Object Oriented Models (for encapsulation and reusability) [136]. However, none of these approaches are of our interest with respect to our research scope. Therefore, we used key-value pairs as the most simple data structure for modeling contextual information, which is also frequently used in distributed service-oriented environment [136].

Using this ontology model, we specify the contextual events and their corresponding application logic. It is used by the CS and DS to react appropriately, when a contextual event happens. The explanation of the ontology model is out of the scope of this thesis.

6.1.2 Data Model Service

The data model service (DMS) is responsible to keep the *data model* and their values. The data model is shared between the orchestration and the decision services. Each service plan has a data model which consists of all the configuration parameters which are used within its orchestration patterns. For instance, the data model of the VsM application consists of t_1,t_2,m_1,m_2, which are shown in Fig. *6-2*. The data model values are used by the DS and OS. To avoid inconsistency of the data, the values of the data model can be changed only by the DS through the $write$ interface, and the OS can only read the values of the data model through the $read$ interface.

6.1.3 Decision Service

The decision service (DS) updates the values of the data model, based on the contextual events and corresponding firing decision rules. The DS needs to be notified about the contextual changes to make decisions [36]. This information is obtained from the CS, as explained before. Since contextual changes can happen any time during the orchestration execution, the decision service needs to be able to receive this information during execution time. As such, the DS supports both synchronous and asynchronous communication with OS and the CS. By synchronous and asynchronous communication we mean pulling and pushing data which are shown in Fig. *6-3*.

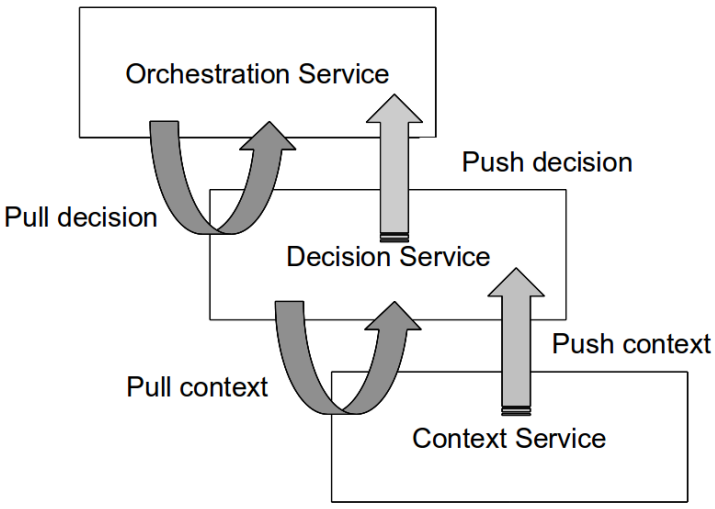

Figure 6-3 : How to pull and push context and decision

For pushing data, the CS should be able to push a contextual event to

the DS and accordingly the DS should be able to push the corresponding decision (an updated value in the data model instance) to the OS. Pushing data is done through publish-subscribe pattern. For pulling data, the DS might need more contextual information and therefore, it should be able to pull context from the CS. Moreover, the OS might need more data model values and therefore, it should be able to pull decision (data model values) from the DS. To provide the puling and pushing data features, the DS makes use of two internal processes shown in Fig. *6-4* and Fig. *6-5*.

Internal process A is used to push decision to OS (Fig. *6-4*). This process is instantiated whenever a contextual event is received from the CS. Internal process B is used to pull decision by the OS from the DS (Fig. *6-5*). This process is instantiated, whenever a request-response call arrives from the OS through the *decision enquiry* interface.

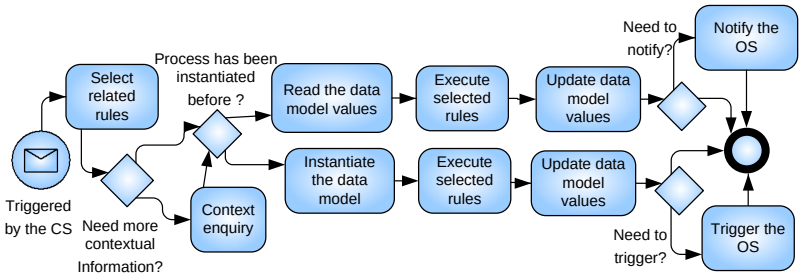

Figure 6-4 : The internal process A of DS to push decision to OS

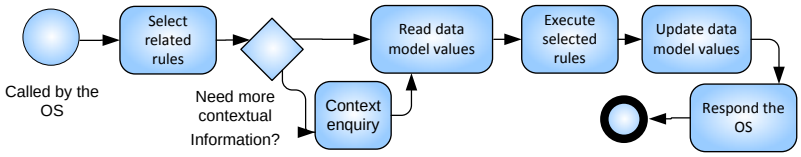

Figure 6-5 : The internal process B of DS to enable the OS to pull decision

Internal process A implements the two interfaces of the DS: *trigger process* interface to instantiate an instance of orchestration pattern through the OS and *notify process* interface to notify a running orchestration pattern instance. This internal process is used to choose decision rules to be executed based on the context information coming from the CS. And it might also query the CS for up-to-date information required for evaluating the selected rules. The contextual events and their corresponding rules are related to a specific service plan and its orchestration pattern. For instance, the calendar event is related to the orchestration pattern of the VsM application. Therefore, if a calendar event is received specifying that the care-receiver's blood pressure

should be measured, then the DS selects the related rule, which is r_0 (in Fig. *6-2*) and queries the last measured blood pressure for rule r_0 (in Fig. *6-2*).

Then, the DS either instantiates a data model by creating an instance of that data model and initialize its parameters (if the VsM application is not instantiated before) or retrieves the values of the data model (if an instance of VsM application is instantiated before) through the DMS. This means that, before executing the selected rules, the DS always retrieves the up-to-date values of the corresponding data model.

After executing the rules, the DS updates the data model values by calling the write interface of the DMS. Based on the executed rules, it might not be necessary to notify or to trigger an orchestration pattern instance. For instance, a blood pressure is measured within the last two hours and therefore, based on r_0(in Fig. *6-2*), there is no need to trigger a new instance of the VsM application.

The orchestration pattern instances are executed independent of each other. So it could be possible that we have a running VsM application instance based on a vital-sign measurement and because the measured value is not stored, a new instance of VsM created based on a calendar event. It could lead to sending a reminder to a care-receiver a few seconds after the vital-sign measurement. However, since a measured vital-sign is stored less than a second, the possibility is very low.

In case of triggering an orchestration pattern instance, the DS calls the OS to trigger that instance. The OS reads the values of the data model from DMS and also subscribes to the DS to receive the notifications related to that orchestration pattern instance through the $subscribe$ interface of the DS. The DS sends only the references of the variables (configuration parameters) that their values are updated. Then the OS reads these new values from DMS.

In case of notifying an orchestration pattern instance, the DS notifies the OS to read the new values of the data model from the DMS. For instance, if a new blood pressure measurement is received by the CS, it triggers the DS and then, the DS updates the data model values and notifies the OS to read the values for the subscribed instance of the VsM application.

For instance, r_6(in Fig. *6-2*) is a reminder rule, which can be executed upon a request coming from the OS exactly before the reminder activity. The DS chooses the rules based on the activity type (e.g., reminder activity). Similar to triggering the process, it might need more contextual information. The DS fires the selected rules, updates the values of the data model and responds to the OS. Then, the OS reads the new values of the data model from the DMS.

6.1.4 Orchestration service

The orchestration service (OS) is responsible to trigger or to notify an orchestration pattern instance based on messages coming from the DS. The OS queries the DS by making use of proper identification mechanisms, then the OS can dispatch incoming messages to proper orchestration pattern instances, and the DS can dispatch incoming messages to proper decision rules and their corresponding data model.

The OS also provides the latest values of the data model to the orchestration patterns by reading them from the DMS. To enable the OS to fulfill these responsibilities, we defined an orchestration template. All the orchestration pattern instances must follow the orchestration template which is shown in Fig. *6-6*. We use BPMN 2.0 [28] notation to illustrate the orchestration template. For instance, the orchestration of VsM application shown in Fig. *6-2*, needs to be modified with respect to this template to be able to use the OS. Fig. *8-4* presented in Section 8.1, shows the modified version of Fig. *6-2*.

As the orchestration template shows, the orchestration instances are triggered by the DS. Immediately after starting the process, two parallel sub-process will start. One is the main orchestration process, and the other is a listener sub-process that can receive the event notifications from the DS.

The listener sub-process has an intermediate message event, which listens to the DS and catches notifications. This listener sub-process is always running until its main sub-process ends. This allows asynchronous notifications (pushing decision) from the DS. Based on the received notifications, the OS reads the new values of the data model from the DMS and then, the listener sub-process returns to intermediate message event, i.e., the listening state ($notified\ by\ the\ DS$ as shown in Fig. *6-6*), to wait for a new notification. The listener and main sub-processes share a local data model instance. Therefore, when the listener sub-process updates the local data model in $read\ the\ data\ model\ values$ activity, the main sub-process also has the updated values.

The main sub-process consists of several activities and decision points. Each activity such as activity A, itself is a sub-process which includes several other activities. An activity is implemented by a call to an application service. The DS is called before and after an activity. As mentioned earlier, each time the DS is called, the local data model will be updated. During the call of the application service, some of these values of the data model might be used.

In the decision points of the main sub-process, there are some conditions, which need to be evaluated based on the values of the data model. Based on their values, one of the alternative paths is selected. Since, these values are assigned by the DS, either in triggering,

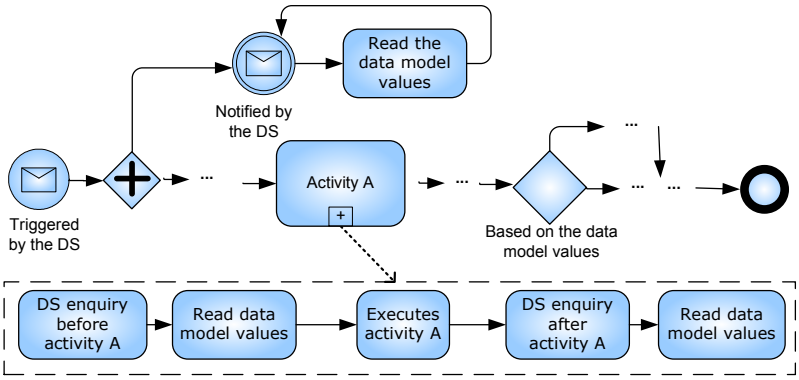

Figure 6-6 : The orchestration template for all the application orchestration patterns

notification, before or after activity call, as opposed to the existing approaches, there is no need to call the DS at the decision points. We assumed that the configuration parameters and their values, which are required by these decision points, are defined in the data model of the orchestration pattern. Therefore, the OS provides the up-to-date values of the data model to the orchestration pattern.

6.2 The Decision Service Template

We define a reference decision service template to implement a decision service in any application domain. The decision service template includes the service interfaces, their input and output messages, and their message exchange patterns to define the behaviour of the decision service. The messages are defined in an abstract level, allowing them to be specialized as required by the application domain. The behaviour and the message exchange patterns remain the same. We use Web Services Description Language (WSDL) to define the abstract interfaces of the decision service. The behaviour of the decision service is modeled by a state diagram. In the following subsections, we explain the interface and the behaviour of the decision service.

6.2.1 The Service Interface

Fig. *6-7* shows the WSDL interface of the decision service in pictorial view, generated using Eclipse Web Tools Platform (WTP) [50]. The WSDL consists of two port types: (a) $Decision\_Process$ and (b) $Decision\_Context$ to interact with the OS and the CS, respectively. The $Decision\_Process$ provides two operations: $subscribe$ and $decisionEnquiry$. The $subscribe$ operation is defined as one-way

operation and is called by the OS as soon as an orchestration pattern instance is triggered and instantiated. This operation takes the instance ID of the new orchestration pattern instance as well as a trigger ID as its input message. The trigger ID is generated by the decision service for each trigger event. Therefore, the decision service knows which data model instance belongs to which orchestration pattern instance.

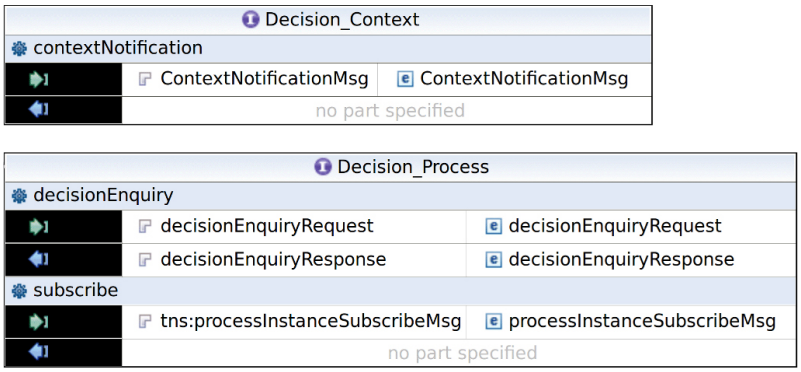

Figure 6-7 : The WSDL interface of the decision service in pictorial view, generated using Eclipse Web Tools Platform (WTP) [50]

The *decisionEnquiry* operation is defined as a request-response operation, which is called by the OS. It receives the orchestration pattern instance ID and the type of activity (before or after which the decision service is called) as input message. Therefore, the decision service is able to select the rules which are related to that specific activity. We assumed that we know all the activity types which can be employed by an orchestration pattern and also their decision rules, which are assigned to before or after of these activity types. The *decisionEnquiry* operation produces the new values of the data model as its output message. These values are also updated in the DMS. The data model, as mentioned before, consists of all the configuration parameters of the orchestration pattern.

The decision service can also notify and trigger an orchestration pattern instance through the OS. The trigger and notify interfaces are implemented by the OS as one-way operation. Therefore, the decision service template does not need any service interface to trigger or to notify the OS.

The *Decision_Context* port type has an one-way operation: *contextNotification*. The decision service subscribes to some specific contextual events based on the rules, which are defined for the deployed orchestration pattern. Whenever one of the events happens, the context service notifies the occurrence of this specific event to the decision service. Regarding the context enquiry, the interface is implemented

by the context service and the decision service can call it to get the latest value of the context.

6.2.2 The Dynamic Behaviour

Fig. *6-8* shows the behaviour of the decision service. When the context service notifies a contextual event or the OS calls the decision service, an instance of the decision service is created. Once an instance of the decision service is created, it either instantiates the data model or retrieves one instance of the data model through the DMS. The data model instantiation is performed to keep track of the up-to-date values of the data model associated to its orchestration pattern instance. The decision service then retrieves the up-to-date values of the data model and enters the reasoning state.

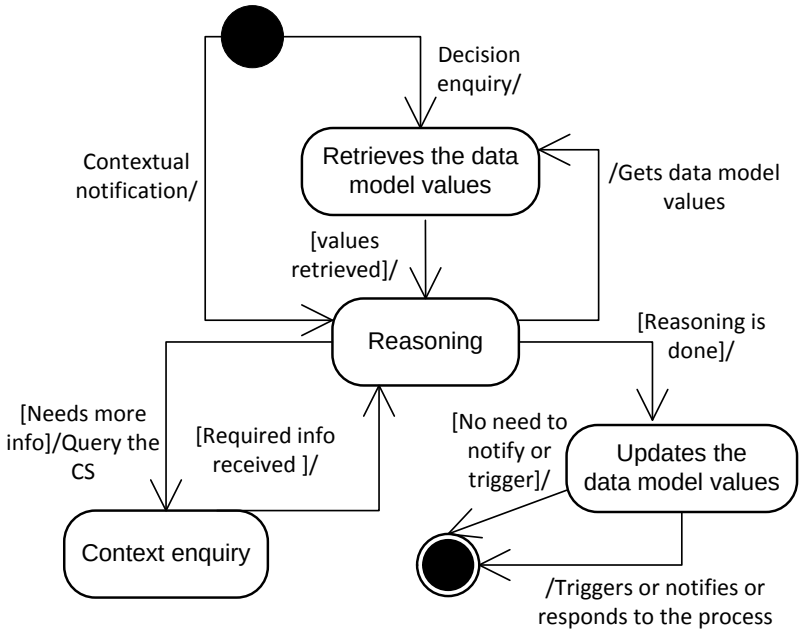

Figure 6-8 : The behaviour model of the decision service

In the reasoning state, if more contextual information is required, the decision service queries the context service and enters the context enquiry state until it receives back response from the context service. As Fig. *6-8* shows, the decision service can be instantiated several times during the execution of one application and the instances are running independent of each other. Therefore, it is possible to handle a contextual event related to a specific application while responding to a decision enquiry coming from the same application. After the reasoning, the decision service updates the data model values in the

DMS. Then, in case of contextual event notification, the decision service, based on the reasoning, may trigger or notify an orchestration pattern instance. If there is no need to trigger or to notify an orchestration pattern instance, the decision service finishes without any action. In case of decision enquiry, the decision service responds to the the OS and then finishes.

Chapter 7

Assumption-based Risk Identification Method

"Living at risk is jumping off the cliff and building your wings on the way down."
— Ray Bradbury

In this chapter we consider service-oriented applications composed of component services provided by different, economically independent service providers. As in all composite applications, the component services are composed and configured to meet requirements for the composite application. However, in a field test of composite service-oriented applications we found that, although the services as actually delivered by the service providers meet their requirements, there is still a mismatch across service providers due to unstated assumptions, and that this mismatch causes an incorrect composite application to be delivered to end-users. Identifying and analyzing these initially unstated assumptions turns requirements engineering for service-oriented applications into risk analysis.

In this chapter, we describe a field test with an experimental service-oriented homecare system, in which unexpected behaviour of the system turned up unstated assumptions about the contributing service providers. We then present an assumptions-driven risk identification method that can help identifying these risks, and we show how we applied this method in the second iteration of the field experiment. The method adapts some techniques from problem frame diagrams to identify relevant assumptions on service providers. The method is informal, and takes the "view from nowhere" in that it does not result in a specification of the component services, but for every component service delivers a set of assumptions that the service must satisfy in order to contribute to the overall system requirements. We end this

chapter with a discussion of generalizability of this method.

This chapter is organized as follows:

Section 7.1 elaborates the problem that we aim to address by introducing the ARM method. Section 7.2 explains the related work and positions the ARM method with respect to them. Section 7.3 describes a homecare system field experiment that showed unexpected behaviours due to unstated assumptions. Section 7.4 describes our assumption-based risk identification method (ARM). Section 7.5 shows how we applied ARM in the second iteration of the field experiment, and the risks that were identified. Section 7.6 discusses the results and lessons learned from the field experiment.

7.1 The Problem Statement

An application service (i.e., composite application) can be built without knowing the internal implementation mechanisms of the services. A composite application composed of component services provided by independent providers, is implemented as a network of actors (including service providers and end-users) that, as a whole, must satisfy application requirements, that in turn are motivated by overall goals.

Each of the contributing service providers makes assumptions about its environment (consisting of the other component service providers, and the end-users) and delivers a service that, if those assumptions were correct, would satisfy the specification of that component service. In practice, these assumptions are mostly unstated, and some of them are incorrect, or at least mutually inconsistent across different service providers.

In our field experiment, these unstated assumptions created unexpected behaviour of the applications, which violated the application requirements. The interactions between component services that made incorrect assumptions about their environment had been unforeseen, and were unwanted.

This problem does not simply go away by specifying in detail what each component service must do, and then matching provided services with this specification. The specifications of component services are often not fully available to the programmer creating the composite application.

Component services are subject to many requirements in addition to those that derive from one particular application. This has influenced their internal implementation decisions, their interpretation of our requirements on their service, and the assumptions they implicitly make

on their environment (i.e. the other component services and the end-users). This leads to a situation in which the service providers and application programmer all believe, mistakenly, that provided services are required services.

The unpredictability of composing services provided by independent providers increases with dynamic service provisioning. In *dynamic service provisioning*, a composite application can be reconfigured. Dynamicity increases the range of possible application behaviours and therefore also increases the range of possible unexpected interactions between component services.

It is important for homecare systems to avoid unexpected behaviour and so requirements engineering for these systems contains a risk assessment. In the literature, different definitions of risks have been given in different domains [144, 81, 49]. We will stay close to the dictionary definition of risk by defining it as "*the possibility of loss or disadvantage to end-users due to composite application behaviour*". The loss or disadvantage to end-users of an application can happen when that application either acts in a way that is not desired by the end-users, or fails to act when it should have responded to environmental changes.

We will *not* assume that this risk is quantifiable. In home care, care-givers are aware of safety risks but cannot quantify them.

If all the assumptions for all possible environmental changes can be stated explicitly during design time, and all provided services can be shown to satisfy the composite application's requirements under these assumptions, there would be no need to do a risk assessment. The first condition is not met in dynamic service provisioning in general, and the second condition is not met because of unstated assumptions and specifications.

7.2 Related Work

Garlan et al. [60, 41] have identified the problem of incorrect or conflicting assumptions for building a system out of existing subsystems. As one of the solutions to alleviate this problem, they have proposed to make these assumption explicit. In our approach, we aim to make the assumptions on the component services as explicit as possible. To do so, we investigate how and why a service provider might interpret the implicit parts of an assumption differently, and how these different interpretations could lead to a risk.

Knowledge Acquisition in automated specification (KAOS) [146] emphasized on identifying obstacles that prevent the achievement of

a goal such as, safety, security, or user-friendliness goals. If these obstacles are unrecognized or underestimated, the requirements on the system and its environment would be inadequate and incomplete. Then the system is exposed to a variety of risks. The obstacle analysis in KAOS is formal and can be applied for any types of goals. While the ARM method is informal and limited to the safety and security risks.

Failure Mode, Effects and Criticality Analysis (FMECA) [27], identifies the risks which would happen if a component fails and prioritizes them based on their severity, frequency of occurrence, and detectability. However, risks of incomplete assumptions are not limited to the failure of one of the component services. In the cases for which our method is intended, component services work properly with respect to their corresponding assumptions, but incomplete or incorrect assumptions causes a risk. This is the core observation that motivated the STPA hazard analysis method [118, 79]. However, the STPA approach is designed for a distributed system with dedicated components and one actor who is responsible for the entire system design, while we have a provisioning platform with no dedicated component services, and we should identify the risks based on incompletely specified services offered by independent service providers.

Argumentation technique combined with the Jackson's problem frame can be used to investigate the security requirements on components of a system and the assumptions behind them [69]. However, ARM emphasizes on identifying the mismatch among several component services due to their unstated assumptions instead of focusing on one component service and its corresponding assumptions.

HAZards and OPerability studies (HAZOP) [88] identifies a set of risks that can arise due to deviation of the system from the intended design specification by a set of parameters and guide words, such as 'more', 'late' and 'no'. The guide words of HAZOP are not directly applicable to identification of risks of incomplete assumptions. Our work is similar to an extended version of HAZOP for programmable electronic systems [125]. However, we tailor risk identification even further to our specific problem domain, as we investigate how requirements of a service provider can affect its corresponding assumptions which are represented as natural-language-like descriptions.

ARM is similar to the RISA method for identifying security requirements in a network of components [113]. However, RISA focuses on security requirements and uses public databases of security vulnerabilities, where we focus on safety and use methods derived from safety engineering. And where RISA extends problem frames with

the Toulmin argumentation technique, we use an informal reasoning approach based geared to sharing the risk assessment with stakeholders who are not computer experts.

Human factors such as end-users exceptional behaviours, have been considered for identifying risks [71]. Similarly, we have investigated the risks of incorrect assumptions on end-users [108]. We consider our proposed method as complementary to the existing risk management approaches to identify new risks. Moreover, we do not talk about risks prioritization since it can be done using the existing approaches.

7.3 Our Field Experiment in the Homecare Domain

We conducted a field experiment at Orbis, a care-institution in the Netherlands [115]. This institution consists of residential blocks where elderly can live and receive care services that are provided by professional care-givers. The aim of this institution is to provide round-the-clock services to their care-receivers and at the same time to enable them to live an independent life as much and as long as possible.

As part of the U-Care (User-tailored Care services platform) project [142], we developed a prototype of an IT-based homecare service provisioning platform [8].

The field experiment has been done in two iterations of two months each, with one month in between to improve existing applications and to add new applications. The problems that we have faced in the first iteration motivated us to design the ARM risk identification method, which we then applied in the second iteration of the field experiment.

Fig. *7-1* shows some of the infrastructure and application services of the U-Care system. It is an instance of the architecture shown in Fig. *2-1*. More details and justification of this architecture has been given elsewhere [7, 8].

In this section, we explain the vital-sign monitoring (VsM) application of the U-care system to motivate the need for the ARM method. In Section 7.4.2, we explain the medication monitoring (MdM) application to validate the ARM method in the second iteration of our field experiment.

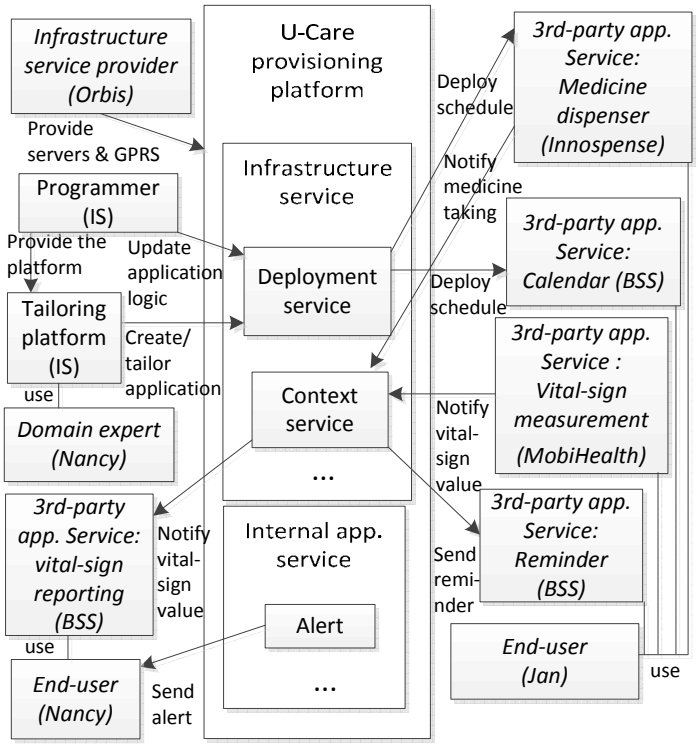

Figure 7-1 : Part of the architecture of the U-Care system.

Fig. *7-1* shows all the component services which are employed by VsM and MdM applications. There are 3rd-party application services which are used by the VsM application: calendar, reminder, vital-sign measurement and vital-sign reporting and medicine dispenser. The medicine dispenser service is only used by the MdM application and will be described later.

The calendar, reminder and vital-sign reporting services are provided by the Biomedical Signals and Systems (BSS) group of the University of Twente [140]. These services are running on end-user Tablet PCs available to the care-givers and care-receivers.

The tailoring platform is provided by the Information System (IS) group of the University of Twente [141]. It is running as an application on end-users Tablet PC available to the care-givers.

If the application logic needs to be updated manually to address unforeseen changes, a programmer of IS modifies the application logic and accordingly, updates the tailoring platform. The tailoring platform, through the deployment service of the provisioning platform, deploys a schedule to the calendar service based on the deployed service plan.

The vital-sign measurement service is provided

by the MobiHealth [106] company. Care-receivers use a vital-sign measurement device at home, which is connected to a server in MobiHealth. The MobiHealth server forwards vital-sign measurement values (e.g., blood pressure), which it receives from the measurement devices, to the context service.

We also show an internal application of the provisioning platform. The alert service sends an alert to a care-giver's PDA. This internal application service and the infrastructure services are running on top of the infrastructure which are provided by Orbis as the care center in our field experiment.

Orbis also provides communication infrastructure for the 3rd-party application services and the PDA or Tablet PC used by the care-givers and care-receivers.

The U-Care system architecture illustrates some of the problems with the composition of services provided by independent providers. We explain this using a scenario in which Jan, a care-receiver, measures his vital signs at home, and in which Nancy is the care-giver responsible for Jan. Nancy creates the vital sign monitoring (VsM) service plan for Jan. The service plan helps Jan to measure his vital-signs (e.g., weight) on time.

After the service plan deployment, Jan can see his schedule for vital-sign measurement on the calendar service running on his Tablet PC. The VsM application starts based on a the scheduled time, and reminds Jan, possibly several times, to measure his vital-signs such as blood pressure. Vital-sign measurement is actually an interaction with MobiHealth, who notifies the provisioning platform of the measurement. If Jan does not measure it on time, or if his vital-signs are not in the normal range, the application sends an alert to Nancy. To monitor whether a vital-sign measurement is not on time, the U-Care system uses the vital-sign reporting service provided by BSS.

This application was tested in the first iteration of the field experiment. The test revealed several problems, of which the following three are illustrative. Fig. *7-2* zooms in on the part of the architecture of Fig. *7-1* that supports the VsM scenario. Each box represents a service which is provided by a service provider mentioned between the parentheses. To explain the problems, the internal implementation components of each service are also shown. Fig. *7-2* shows that MobiHealth gives a vital-sign measurement device and an smart phone to the care-receiver. The smart phone communicates with the measurement device by Bluetooth and forwards the measurement data to a web server. Then the web server of MobiHealth, forwards the data to the context service of Orbis running on an application server (i.e., App server). Orbis stores the data and then forwards the data to

the vital-sign reporting service of BSS. This service is running on an application server that communicates with Nancy's Tablet PC.

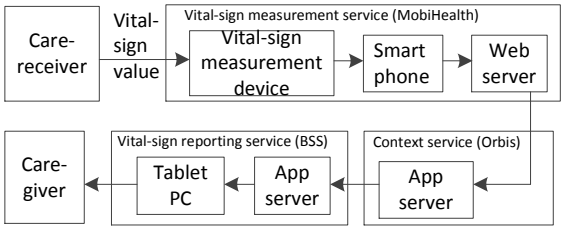

Figure 7-2 : The part of the U-Care architecture supporting the VsM scenario.

We classify the problems into three kinds, so that we can refer to them later.

– *Service availability problem:* The Orbis application server is down for maintenance every night at 3 a.m. Data received in downtime would be lost. However, unexpectedly, some care-receivers measured their blood pressure around 3 a.m.

– *Data transportation problem:* The Bluetooth connection between the measurement device and the smart phone of the care-receivers resends data when the sender does not receive an acknowledgment on time. This turned out to happen sometimes, and then leads to data duplication not discovered by the rest of the network, i.e. all other services did not recognize that data was duplicated.

– *Data storage problem:* All the data must be hashed before transportation among the component services. Due to hashing algorithm used by MobiHealth, the hashed care-receiver's ID (identification number) exceeds the maximum size of care-receiver's ID defined by BSS. Therefore, Orbis faced an error when it forwards the hashed care-receiver's ID to BSS. Because BSS can not store them and sends an error back to Orbis.

These cases of architectural mismatch can be traced by lack of knowledge of each others internal implementations and operational processes, as well as lack of knowledge about how end-users behave. Some implementation decisions are made for good reasons, others are bad decisions, but none of them are under the control of a central coordinator.

7.4 The Proposed Risk Identification Method

In this section we first give an overall description of the method, using the VsM application as an example. We then provide step-by-step instructions on how to apply the method in general.

7.4.1 Overall description

We assume that our risk identification method is used by the *owner* of the composite application. The owner is the actor who specifies the requirements for, and is responsible for the delivery of the composite application. We do not assume that the owner is in charge of any of the component services used to compose the composite application. Therefore, the owner cannot specify or implement any of these services. The owner has the "view from nowhere" because he does not take any of the component service providers' point of view.

The owner is responsible for selecting component service providers, and composing them into a composite application that satisfies his composite application requirements. To discharge of its responsibility, the owner must be able to give an argument of the form:

> If service providers $S_1, ..., S_n$ behave like this: $A_1, ..., A_n$, respectively, then the composite application satisfies its requirements.

This is called a frame concern by Jackson [82], but because we are reasoning about components services and not about problem domains, we call it a *contribution argument*. Note that instead of requirements on service provider S_i, the contribution argument makes *assumptions* about the service provider meeting his requirements. We therefore have two types of assumptions: assumptions made by service providers about the environment of their service, and assumptions made by the application owner ($A_1, ..., A_n$) as part of his contribution argument. The assumptions made by the owner take the form of descriptions of the behaviour of the components as inferred or desired by the owner.

Fig. *7-3* extends the architecture diagram of Fig. *7-2* with annotations that illustrate parts of the contribution argument. We borrow from problem frame diagrams the notation to represent requirements in a dashed ellipse, connected by dashed lines to the actors who are the subject of the requirements. Fig. *7-3* contains the requirement R1 on the care-giver and care-receiver:

R1 The care-receiver shall provide vital-sign values to the care-giver according to the service plan for vital-sign measurement. The care-giver shall respond to situations in which the care-receiver does not follow the service plan, and situations where the measurements are outside the safe range as stated in the service plan.

This requirement can be fulfilled if the care-giver is permanently present at the care-receiver's location. The U-care system is introduced to fulfill the requirement even when the care-giver is not permanently present.

The owner of the vital-sign monitoring (VsM) application service has designed an architecture of independent service providers, and translated requirement R1 into a set of assumptions about services provided by these service providers, such that these assumptions jointly satisfy R1. The contribution argument now becomes

> If the service providers in the architecture of Fig. *7-3* satisfy the assumptions listed in Fig. *7-3*, the composite application satisfies R1.

Because the owner is responsible for more U-Care applications, this architecture may contain components that are not optimal for implementing R1. For example, a simpler architecture from the standpoint of implementing R1 only, would be to have the vital-sign reporting service be provided by MobiHealth instead of by BSS. However, there are other requirements that call for an independent vital-sign reporting service, which motivates the choice made in Fig. *7-3*.

Given an architecture, assumptions are placed on its components so that the contribution argument can be given. This corresponds to adding "breadcrumbs" (assumptions) in terms of Seater & Jackson's method to transform a requirement into a machine specification [131].

The assumptions on service providers are assumptions on *interfaces* of the service providers. An assumption listed under the box representing the service provider is about its interface represented by the line (e.g., in Fig. *7-3*, assumption$_{i2}$ is on "interface i2" of "vital-sign measurement device"). Assumptions made by the composite application owner are not specifications to be implemented by a service provider. Rather, the owner has to check if the service provider has the capability to satisfy the assumptions made on it. The service provider has a range of possible behaviours that we call its *capabilities.* An actor uses its capabilities to provide services, and usually does so in the context of different composite applications. For example, MobiHealth provides services in many other contexts in addition to the U-care

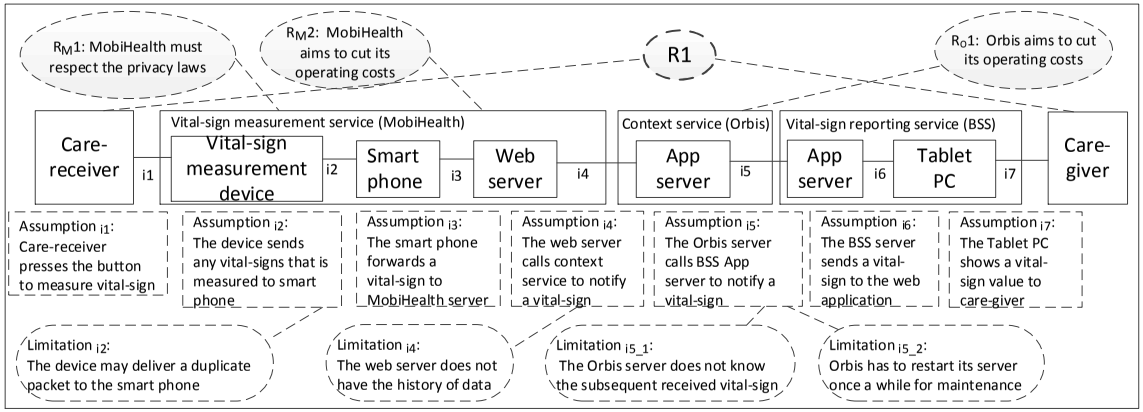

Figure 7-3 : The network for requirement 1 to be satisfied by vital-sign monitoring (VsM application).

context, and does so with the same services as those contributed in the U-Care system.

Every capability implies a *limitation*: every range of possible behaviours implies that there are behaviours *not* in this range. Fig. *7-3* lists some limitations of capabilities by a dashed oval box, connected to the assumption that they qualify. The assumption states a capability that the actor is assumed to have, and the limitation is a limitation of the actor's capabilities that causes it to fail to meet the assumption. Note that the actual capabilities of an actor are nowhere stated; they are unknown to the owner of the composite application. Rather, the owner states an assumption about these capabilities. A limitation shown in the diagram states a property of the capability that the actor actually has. It tells us why this actor, with this limitation, fails to meet the assumption. The diagram in Fig. *7-3* shows four limitations, which are four reasons why this architecture with these service providers, fails to meet requirement R1. In other words, the diagram shows four risks of this particular network.

Mitigating these risks involves a choice between accepting the risks (occasional failure on the system), transferring them (taking out insurance against failure), avoiding them (dropping requirement R1), removing it (replacing a service by one that satisfies the assumptions on it) or compensating it (changing the capability of some service provider so that the limitation stops being a case for failing to meet R1). Risk mitigation falls outside of the scope of this thesis.

Fig. *7-3* also lists the reasons why some actors have some limitations in their capabilities. The diagram shows two requirements on MobiHealth and one on Orbis that are extraneous to the U-Care project. The actors are subject to these requirements independently from whether or not they participate in U-Care.

These requirements are themselves motivated by higher-level goals, namely privacy laws in the homecare domain [32] and business goals (cost-effectiveness). The reasons explain why some actors made some implementation choices, that caused the limitations noted in the Fig. *7-3*. They also make clear that these choices will not be changed just to satisfy a U-Care requirement.

Finally, note that the three requirements listed for MobiHealth and Orbis are context-free in the sense that they refer to one actor only. This explains why these requirements are not necessarily mentioned in discussions with other actors or with the owner of the application.

We next classify the limitations that we have found in the first iteration of our field experiment into three kinds, illustrated earlier by three problems.

– *Service availability limitation:* A service provider has a limitation on its capability to a service always running.

– *Data transportation limitation:* A service provider has a limitation on its data transportation capability, such data transportation assumptions made by the composite application owner are not satisfied. (This assumption could be: no later than delivery, no earlier than delivery, at most once delivery, exactly once delivery etc.)

– *Data storage limitation:* The capability of a service to add, store or delete data may differ from that assumed by the owner of the composite application.

We claim that these are all the limitations that the owner should look for when doing a risk assessment.

7.4.2 Step-by-step description of ARM

Our assumption-based risk assessment method ARM consists of three steps. We assume that the requirements are known and that an architecture for the composite application has been designed in which service providers provide the component services. We now identify relevant assumptions, and search for risks that the requirement will not be satisfied.

1. *Translate the main requirements into assumptions on the interfaces of the network actors.* This is done by constructing a contribution argument in which the assumptions on the interfaces of service providers are listed, needed for ensuring that the composite application satisfies its requirement. In our experience we find this step the most difficult one. In our field experiment, the programmer with the help of the service plan (given by a nurse), and service specification and end-user manual (given by the service providers) writes down these assumptions. How complete and accurate are they? This is a difficult question and its answer is outside of the scope of this thesis.

2. *Explore the capabilities and limitations of each service provider to satisfy the assumptions on its interface.* That is, we try to identify capabilities that a service provider should have in order to satisfy the assumptions on its interface. These capabilities are in correspondence with the three types of limitations which we have faced in the first iteration of our field experiment. They are mainly

related to how a service provider manipulates data.
Here are sets of questions that we have found useful to ask about each service provider, in order to assess whether it has the assumed capabilities.

(a) *Service availability questions.* What are the operating systems of the internal components used by the provided service? How (and how often) are they maintained? What are the limitations of employed software or hardware? For instance: do they need to restart for maintenance and if yes, how long does it take and how often? How will a software or hardware component be updated (e.g., on-the-fly)? Do they have a second power supply? How long can they continue on the second power supply? If they use a battery, how long can they go on battery power?

(b) *Data transportation questions.* How are the internal components connected to each other? How reliable are services and the interconnections (e.g., error handling, packet lost, duplicate packet, ..)? What are the limitations of employed networks and networks protocols? For instance: do they guarantee the messages will be transfered? Be sent maximum once? Do they send an acknowledge? What happens if before receiving an acknowledge the connection is lost?

(c) *Data storage questions.* How are the internal data stored? What are the primary or foreign keys? How are these data added or edited or deleted? What are the limitations for storing or internal transportation the data values? For instance: does the provider have the permission to store the data? How long can the provider keep the data? What is the minimum level of encryption for storing and transporting the data? Is data hashing sufficient? Can the provider find out who is the owner of the data? Can the provider make a distinction between data based on their owner?

Questions like these can be answered by inspecting the contract with a service provider, or, if more detailed information is needed, by inspecting technical documentation provided by a service provider or by interviewing experts inside the service provider organization.
The answers for these questions might be motivated by the

extraneous requirements on the service providers.

3. *Explore in which way each limitation identified in the previous step could cause the service provider fail to meet an assumption.*
   One way to do this is to change keywords in the capability descriptions in a HAZOP-like manner [88] [125]. The assumptions are written in natural language, and so variations in these assumptions can be derived by adding or removing some words (e.g., adverb) in the assumptions.
   For the VsM application, limitation$_{i2}$ and limitation$_{i5_1}$ could be changed into corresponding assumptions as follows:

   – Assumption$_{i2'}$: The device *might* send any vital-signs that is measured to smart phone *more than once*.

   – Assumption$_{i5'}$: The Orbis server *might* call BSS web service to notify a vital-sign *more than once*.

   By knowing these two affected assumptions, we could have identified the *Data transportation problem* of duplication data before the field experiment. Since it is not a serious risk, we could inform care-givers for such application behaviour and asking them to take care of that (transferring the risk).
   As another example, the limitation$_{i4}$ and limitation$_{i5_2}$ could be changed into corresponding assumptions as follows:

   – Assumption$_{i4'}$: The web server calls context service to notify a vital-sign *and delete the data*.

   – Assumption$_{i5"}$: The Orbis server calls BSS App server to notify a vital-sign *not during the maintenance*.

   By knowing these two affected assumptions, we could have identified the *Service availability problem* and the risk of $losing\ vital\text{-}sign\ measurement\ data$ did not occur. It is a serious risk and either Orbis has to remove limitation$_{i5_2}$ or MobiHealth has to remove limitation$_{i4}$.

7.5 Applying ARM to Our Case

In the second iteration of our field test, we applied the ARM method to the medication monitoring (MdM) application scenario. In this

scenario, the care-receiver uses medicines made available by a medicine dispenser provided by the Innospense company [78]. As in the first iteration, we use ”Nancy” as short-hand for the care-giver, and ”Jan” for the care-receiver. Nancy creates a service plan, using the tailoring platform, to help Jan to take his medication on time (see Fig. *7-1*). Similar to the VsM application, if Jan does not take his medication, the MdM application will send a reminder to him, possibly several times (shown on his Tablet PC). If Jan has not taken his medication after a tailored number of reminders, the application will send an alert to Nancy (shown on her PDA).

Fig. *7-4* zooms in one the part of the architecture of Fig. *7-1* that supports the MdM application scenario. It shows three sub-networks. In sub-network (1), the medicine dispenser, which is located at Jan's home, communicates with the web portal of Innospense and forwards the timestamp of Jan taking his medication. The web portal then forwards the timestamp to the context service running on an application server of Orbis.

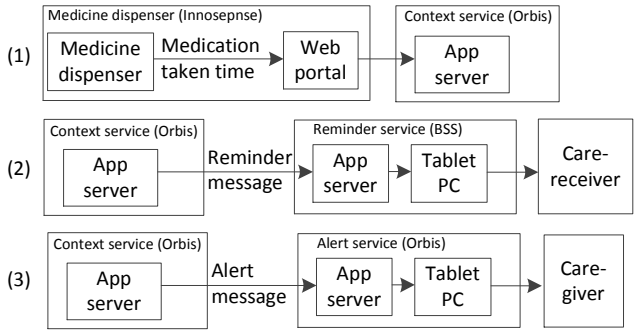

Figure 7-4 : Part of the U-Care architecture supporting the MdM scenario.

In sub-network (2), the context server checks at the scheduled time for medication if it has recently received the timestamp from Jan's medicine dispenser. If no timestamp has been received, the context service sends a context-aware reminder message (e.g., if Jan is outside his home, the reminder message would be: “please go home and take your medicine”) to the reminder service of BSS. This service is running on an application server that communicates with Jan's Tablet PC.

In sub-network (3), if the context service has not received a timestamp from Innospense after having sent several reminder messages, it will send a context-aware alert message (e.g. “Jan is outside his home and has not taken his medicine yet”) to the alert service. This service runs on the application server of Orbis that forwards the alert to Nancy's PDA.

The MdM application must satisfy requirement R2 on the care-giver

and care-receiver:

R2 The care-receiver shall take a medicine from the dispenser at the scheduled time according to his service plan. The care-giver shall respond to situations in which the care-receiver does not follow his service plan.

Fig. *7-5* contains R2 and extends the architecture diagram of Fig. *7-4*. This figure is explained through the steps of the ARM method.

Step 1: *Translate the main requirements into assumptions on the interfaces of the network actors.*
Based on the service plan, the programmer as the owner of the MdM application, drew the network of actors for the MdM application according to its service plan. As elaborated in a previous paper [8], the service plan contains a BPMN (Business process modeling language)-like part that shows the combination of the component services.

Then, the programmer wrote down the assumptions on capabilities of each actor. In this case, the end-user manual of Innospense turned out to be useful, since it specified the external behaviour of the medicine dispenser.

As the figure shows, if the service providers satisfy assumptions$_{i8,i9,i10,i11,i12}$, Jan will receive a reminder if he forgot to take his medicine. Besides, if the service providers satisfy assumptions$_{i8,i9,i13,i14,i15}$, Nancy will receive an alert message if Jan does not take his medicine after several reminder messages. Together, these assumptions fulfill requirement R2 even when the care-giver is not permanently present.

Step 2: *Explore the capabilities and limitations of each service provider to satisfy the assumptions on its interface.*
As an example, based on interviews with the service providers we identified this requirement for Innospense:

R$_I$1 Innospense aims to reduce its communication costs between devices.

This requirement in turn motivated Innospense to make some implementation decisions. We uncovered these decisions by asking the three *Service availability, Data transportation, and Data storage questions*. Innospense limits the communication between its web portal (located in the company) and the medicine dispenser device (located at home) to some specific time granularity. As such, limitation$_{i8}$ is

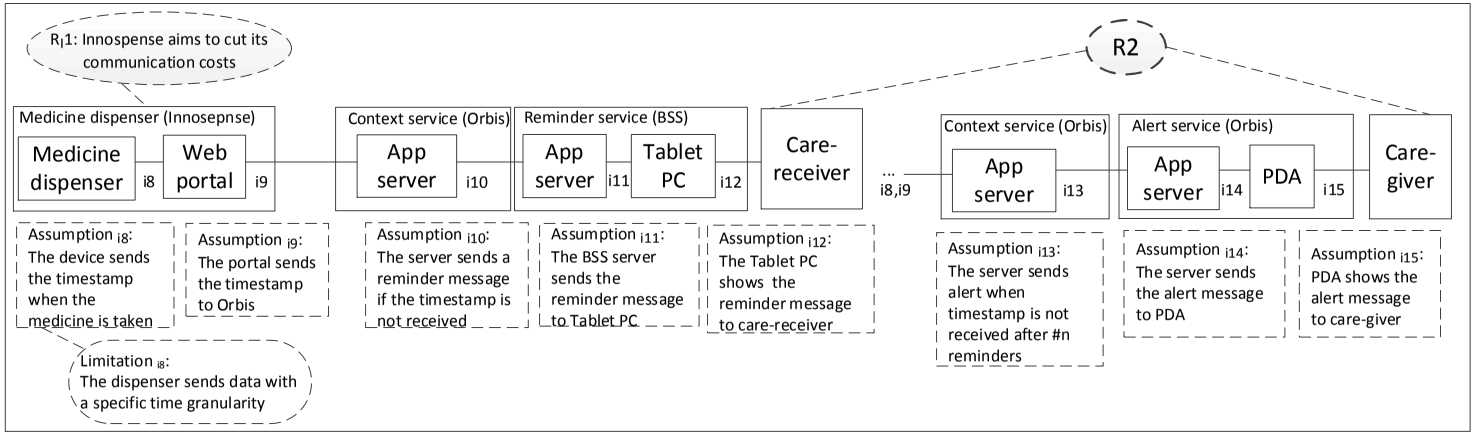

Figure 7-5 : The network for requirement 2 to be satisfied by medication monitoring (MdM) application.

imposed.

Step 3: *Explore in which way each limitation identified in the previous step could cause the service provider fail to meet an assumption.*
To continue our example, we found the following impact of limitation$_{i8}$ on assumptions$_{i8}$:

– Assumption$_{i8'}$: The device sends the timestamp *with a maximum delay of m minutes after* the medicine is taken.

Based on the affected assumption, we identified the following scenarios and risks:

Risk 1 for assumption$_{i8'}$: we can think of a scenario in which Jan receives a reminder even though he has taken his medication. This happens when the scheduled time for taking the medicine expires before the timestamp is received by the context service, due to the delay introduced by the medicine dispenser.

Risk 2 for assumption$_{i8'}$: we can think of a scenario in which Nancy receives an alert even though Jan has taken his medication. This happens when the scheduled time for sending an alert is reached (after sending the last reminder and) before the timestamp is received by the context service, due to the delay introduced by the medicine dispenser.

There are other assumptions, limitations and risks in addition to those mentioned above, but these have been omitted for brevity. We indeed completed the risk assessment for the MdM application before the second iteration of our field test. Therefore, the provisioning of the MdM application was more reliable than the VsM application in the first iteration of our field test.

7.6 Discussion and Future Work

We have presented and illustrated a method to identify risks in a decentralized system that is composed of component services provided by independent providers. Each component service provider is responsible for the way it provides its service, and the owner of the composite application cannot demand a component service provider to provision a dedicated service. In this situation, the owner of the composite service does not *specify* component services, but must make *assumptions* about them, that reflect agreements made with every component service provider.

The end-users are not computer experts, which limits the kind of

assumptions we can make on end-users. At the same time our system is safety-critical because a failure to meet system requirements may harm end-users. Riskiness is increased by dynamicity in the form of adaptability, tailorability and evolvability of the system.

The ARM method starts out with the requirements on a composite application and the component service network used to meet these requirements. It proceeds by listing the assumptions about each component service, checking whether the component service provider really has the capability to meet these assumptions. Then, it explores the impact of any limitations of these capabilities for the ability of each component service provider to meet its assumptions, and hence of the service network to meet the overall requirement. Any risks identified this way are managed by accepting, avoiding, transferring or reducing the risk. Mitigation is typically done in agreement with the stakeholders but is outside the scope of the ARM method.

We have developed and used the method in a field experiment of a homecare system. This test showed that the method can be used and delivers useful results in this case. A possible threat to internal validity could be that the risk identification actually was successful, not because ARM was used, but because we were aware of the risks in some other way already. However, this was not the case: We were not aware of the risks identified in between the two iterations before we used the ARM method.

An important threat to external validity is that perhaps we were able to use this method in this project, but that this is not repeatable in other projects, neither by other people nor by us. Usability by other people in other projects must be shown in future experiments, by giving this method to other risk assessors in similar projects. The method is arguably repeatable in similar projects with decentralized service networks because the method explicitly assumes such decentralization and provides means to represent it and reason about it. Moreover, the method might only be feasible for human analysts rather than for automated tools, as the analysis rules and representations are hard to formalize.

We specify the assumptions in a natural-language-like description. Therefore, unlike HAZOP, we are not limited to a set of predefined guide words. Instead, we can more informally investigate the variation of the assumptions by either adding or removing any words.

The ARM method can be applied as soon as the design of the composite application has completed. Risks can be identified before the application is built or the component services have been implemented. ARM can therefore be used in an early stage of application development when mistakes can still be corrected with relatively little cost.

After identifying a risk, we need to reach an agreement among the service providers on how to prevent or to mitigate that identified risk. Therefore, one or several actors should pay the cost and to some degree compromise their requirements. To do so, the requirements of actors should be prioritized. Some of them, for instance the privacy law, are compulsory and the service providers must comply. Nevertheless, some others such as the goal of reducing cost can be negotiated. We can even inform the stakeholders, for instance the care-givers, about a risk and if it is acceptable, there would be no need to (re)plan the application behaviour. Instead, we plan end-users behaviour by making them aware of the risk in advance.

Another challenge that we have found in our pilot study is that after introducing the homecare applications, the end-users change their behaviour because of the new possibilities. For instance, Jan has not to wait until Nancy comes to his apartment to measure his blood pressure. Therefore, he often measures his blood pressure earlier than scheduled time and goes out to meet his friends. Predicting these type of end-user behaviour as *what if* scenarios [147] is not a straightforward task at the design time.

Chapter 8

Experimental Prototype

"A good idea is about ten percent and implementation and hard work, and luck is 90 percent."
— Guy Kawasaki

In this chapter, we discuss the prototype implementation of the proposed dynamic homecare service provisioning (DHSP) platform. The prototype is a proof of concept to be validated using a near real-life setting. As explained in Chapter 5, the DHSP platform consists of two types of services: *infrastructure services* and *application services*. To explain the DHSP implementation, we should explain how these two types of services have been implemented including their message formats (to exchange data) and database structure (to store data). Since all the application services were provided through the SOAP [150] protocol, we implemented both types of services as web services using WSDL [151] specification. Using web service enables the DHSP platform to be decentralized, i.e., all the infrastructure and application services can be hosted on different physical servers regardless of their geographical locations as long as these services can communicate with each other through the SOAP protocol. Later in the field test, we chose a deployment architecture which suits the involved stakeholders' requirements. For instance, the application service providers hosted their services on their servers within their organizations.

This chapter is organized as follows: Section 8.1 explains how we implemented the infrastructure services including the decision service (explained in Chapter 6). Section 8.2 explains how we implemented the application services as the implementation of the SBBs (explained in Section 5.1). Section 8.3 discusses the structure of the database used by the infrastructure and application services in our prototype. Section 8.4 explains our deployment architecture and the reasons behind the deployment choices which we have made in our field test.

8.1 Infrastructure Services

In this section, we explain how we specialized and then implemented the logical architecture introduced in Section 6.1. We use the VsM application for blood pressure monitoring (explained in Section 5.1.3) as an example during our explanation. We have employed several tools and technologies such as process and rule engines which can be installed either on one or separate physical servers due the decentralized feature of the DHSP platform.

Fig. *8-1* shows how the infrastructure services, namely OS (orchestration service), DS (decision service), CS (context service), and DMS (data model service) were implemented on top of the process and rule engines. We used WebSphere Lombardi Edition [74] and WeSphere ILOG [75] as process and rule engines, respectively. The implementation of application services including the application server will be explained in more details in next section. However, we also need to show them in this section to explain the infrastructure services.

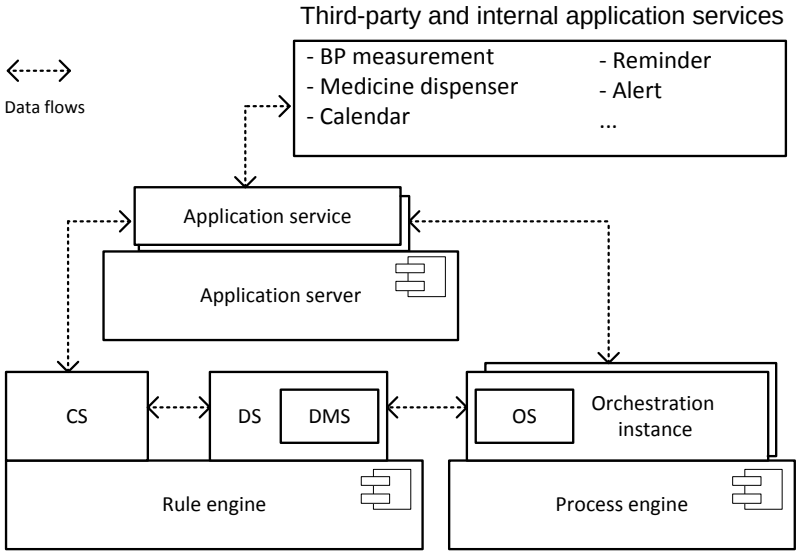

Figure 8-1 : The implementation of the infrastructure services (shown in Fig. 6-1)

Regarding the infrastructure services, the DS and CS were implemented on top of the rule engine, since, both of them make use of several decision rules. The DMS (data model service) was implemented as part of the decision service. Therefore, the decision service can update the data model values locally and the OS (orchestration service) can read the data model values through the decision service. Moreover, the OS was implemented as part of the orchestration instances of our

homecare applications. This means that the OS template (shown in Fig. *6-6*) is implemented by the orchestration instances. Therefore the orchestration instances could be triggered or notified by the DS, or they could query the DS. The orchestration patterns themselves were implemented on top of the WebSphere Lombardi Edition as the processes engine.

To clarify our explanation, we repeat the service plan example here in Fig. *8-2*. The rest of this section is explained with respect to this service plan example.

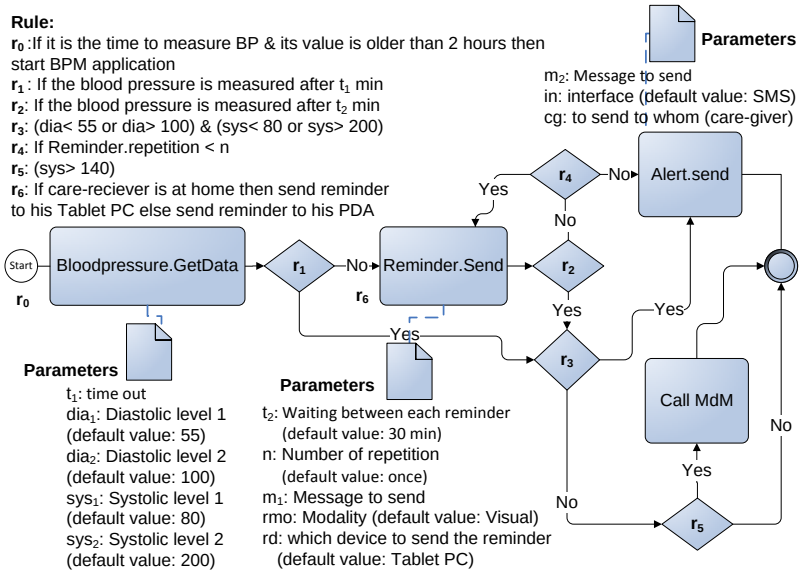

Figure 8-2 : The service plan of VsM application from the programmer's point of view

Fig. *8-3* shows how the infrastructure services interact with each other to implement the VsM application for blood pressure monitoring. As explained above, the OS was implemented as part of the orchestration of VsM application.

The calendar service notifies the context service about a new blood pressure measurement event and then, the context service notifies the decision service by a calendar event. The decision service has subscribed for this event during the VsM application deployment. Because of r_0 in Fig. *8-2*, the decision service needs to query the context service to get the latest measured blood pressure value. If it was not measured, for example within the last two hours, the decision service triggers an instance of VsM application. The instance of VsM application subscribes to the decision service for each event which is used by the orchestration pattern of the VsM application. Before the

reminder activity, the VsM application calls the decision service to execute the related rules. Since, r_6 in Fig. *8-2*, is assigned to be executed before reminder activity, the decision service queries the context service to get the location of the care-receiver, which is needed by this rule. After the reminder activity, the decision service is called to check whether it reaches to the maximum number of reminder repetition or not. Whenever a new blood pressure value is arrived, the decision service executes r_3 in Fig. *8-2*, and notifies the VsM application by the new values of the data model. The VsM application immediately interrupts the orchestration and jumps to the related decision point.

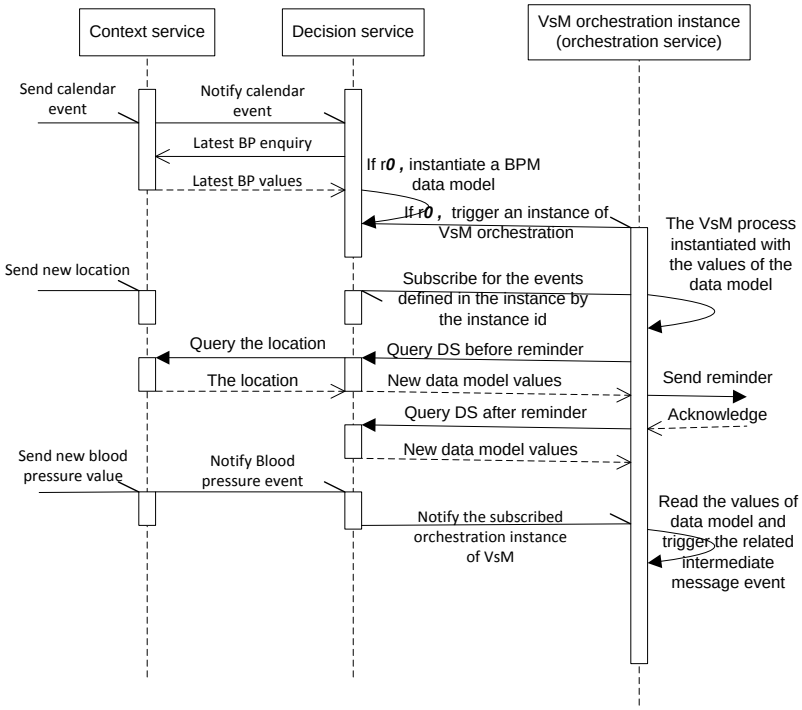

Figure 8-3 : The Interaction of the infrastructure services for the VsM application

The VsM orchestration pattern, which was implemented on WebSphere Lombardi, is shown in Fig. *8-4*. In compliance with the generic process template (shown in Fig. *6-6*), it has a start event and parallel sub-process to catch all the triggers and notifications coming from the decision service, respectively. The start event, which implements the trigger interface of the OS, instantiates (based on trigger event) the VsM orchestration pattern. Then the OS subscribes to the decision service for later notifications used by that orchestration pattern instance. The parallel sub-process, which implements the notify

interface of the OS, reads the values of the data model from the DMS and accordingly, it might post a message to the event manager of Lombardi.

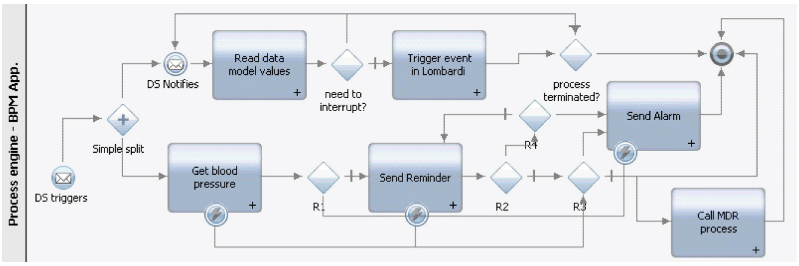

Figure 8-4 : The VsM orchestration pattern on Lombardi in compliance with the generic process template (shown in Fig. 6-6)

This message can be received by all the intermediate message events which are attached to the activities. This enables the orchestration pattern of VsM application to immediately interrupt the execution and to jump to a specific point of the orchestration pattern. For instance, by receiving a new blood pressure measurement, the VsM application jumps to the decision point of r_3 in Fig. *8-2*.

To call decision enquiry before or after the activity, all the activities have an internal process. The internal process of the reminder activity, which implements the enquiry interface of the OS before and after sending the reminder, is shown in Fig. *8-5*. Based on the values of the data model, after the decision enquiry, the process may either choose PDA or Tablet PC to send the reminder.

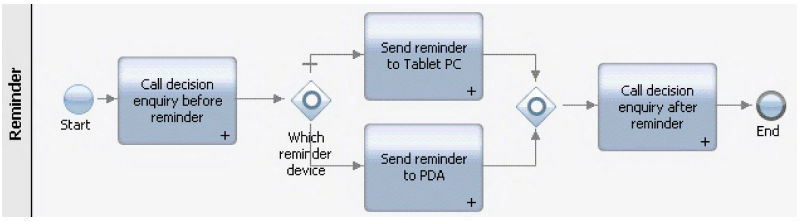

Figure 8-5 : The internal process of the reminder activity

8.1.1 Message Format

Based on our implementation of the infrastructure services, there are two message exchanges (and message formats) among the infrastructure services: (1) between the CS and the DS, and (2) between the DS (including the DMS) and the OS. Since the DS is involved in both message exchanges, we explain the specification of the DS that covers both message exchanges and their used message formats. As we explained, we used SOAP and WSDL to specify our infrastructure and

application services. Listing 8.1 shows the WSDL specification of the DS.

```
<wsdl:message name="decisionEnquiryRequest">
    <wsdl:part name="OrchestrationInstanceID" element="prov:orchestrationInstanceID"> </
    wsdl:part>
    <wsdl:part name="activityTypes" element="prov:ActivityTypes"> </wsdl:part>
    <wsdl:part name="before  after" element="prov:Before  After"> </wsdl:part>
</wsdl:message>

<wsdl:message name="decisionEnquiryResponse">
    <wsdl:part name="dataModel" element="prov:DataModel"> </wsdl:part>
</wsdl:message>

<wsdl:message name="contextNotificationMsg">
    <wsdl:part name="contextualEvent" element="prov:ContextEvent"> </wsdl:part>
</wsdl:message>

<wsdl:message name="subscribeMsg">
    <wsdl:part name="OrchestrationPatternID" element="prov:orchestrationPatternID"> </
    wsdl:part>
    <wsdl:part name="triggerId" element="prov:TriggerId"> </wsdl:part>
</wsdl:message>

<wsdl:porttype name="Decision  Process">
      <wsdl:operation name="subscribe">
        <wsdl:input message="tns:subscribeMsg"> </wsdl:input>
      </wsdl:operation>

     <wsdl:operation name="decisionEnquiry">
   <wsdl:input message="tns:decisionEnquiryRequest"> </wsdl:input>
        <wsdl:output message="tns:decisionEnquiryResponse"> </wsdl:output>
 </wsdl:operation>
   </wsdl:porttype>

<wsdl:porttype name="Decision_Context">
      <wsdl:operation name="contextNotification">
        <wsdl:input message="tns:contextNotificationMsg"></wsdl:input>
      </wsdl:operation>
</wsdl:porttype>
```

Listing 8.1 : DS interface in WSDL

The message *contextNoticationMsg* is exchanged between the CS and DS to notify the contextual event. In our application scenario the contextual events could be either a new vital-sign measurement or medication intake. Therefore, we define a complex type (*contextualEvent*) as shows in Listing 8.2, to represent all the required event types. This complex type consists of three elements:

– *EventType*: It is an enumerated data type to restrict the number of event types. It could be either a vital-sign type or medication event. We have different vital-sign types which will be explained as part of the application services in next section.

– *EventValue*: We convert all of the vital-sign values and medication intake information to *string* format in order to have a unique message format between CS and DS.

– *TimeStamp*: It shows the time that its corresponding event happens. We used a *string* format for instance, "08-04-2012 13:52:40".

The message *decisionEnquiryRequest* and *decisionEnquiryResponse* are exchanged between the DS and OS to update the data model values of the OS. The *decisionEnquiryRequest* has *OrchestrationInstanceID* which is created when the DS triggers an orchestration pattern through the OS. This instance ID is unique for each homecare application (and its data model) instance and stored by the DS and OS as part of the data model. The instance ID is used by the DS and OS to retrieve the correct data model instance in upcoming interaction.

```
<xs:complexType name="contextualEvent">
  <xs:sequence>
    <xs:element name="EventlType" type="prov:eventType"/>
    <xs:element name="EventValue" type="xsd:string"/>
    <xs:element name="TimeStamp" type="xsd:string"/>
  </xs:sequence>
</xs:complexType>

<simpleType name="eventType">
  <restriction base="xsd:string">
    <enumeration value="bood_pressure"/>
    <enumeration value="weight"/>
    <enumeration value="oxygen_saturation"/>
    <enumeration value="medication"/>
  </restriction>
</simpleType>
```

Listing 8.2 : Data schema of the context event used by the *contextNoticationMsg* message exchanged between the CS and DS.

In order to select the decision rules to be executed, the OS also informs the DS that before or after which activity the inquiry is performed. We have defined an enumerated data type (*activity_ type*) which has several *string* activity types such as "*reminder*" and "*alert*". The *before_ after* is also an enumerated data type with two string options: "*before*" and "*after*". Using these two data types, the OS can inform the DS that from where (within an orchestration pattern) the inquiry is performed.

The *decisionEnquiryResponse* message includes the decision and the updated data model values since we implemented the DMS as part of the DS. Moreover, the decision is actually the updated values of a data model instance. Therefore, *decisionEnquiryResponse* message format is the same as data model. Listing 8.3 shows the data model schema which is used as the element of the *decisionEnquiryResponse* message.

```
<xs:complexType name="dataModel">
  <xs:sequence>

<xs:element name="start_process" type="xsd:boolean">
<xs:element name="send_reminder" type="xsd:boolean">
<xs:element name="send_first_reminder" type="xsd:boolean">
<xs:element name="wait_before_first_reminder" type="xsd:int">
<xs:element name="reminder_repetition" type="xsd:int">
<xs:element name="wait_before_reminder_repetition" type="xsd:int">
<xs:element name="reminder_message" type="xsd:string">
<xs:element name="reminder_modality" type="prov:modalityType">
<xs:element name="reminder_binding_port" type="xsd:string">
<xs:element name="send_alert" type="xsd:boolean">
```

```
<xs:element name="alert_message" type="xsd:string">
<xs:element name="alert_interface" type="prov:interfaceType">
<xs:element name="alert_binding_port" type="xsd:string">
<xs:element name="call_medicine_dispenser" type="xsd:boolean">
<xs:element name="care_giver_id" type="xsd:string">
<xs:element name="orchestration\_instance\_id" type="xsd:int">
<xs:element name="service\_plan\_id" type="xsd:int">

</xs:sequence>
</xs:complexType>
```

Listing 8.3 : Data model used by the *decisionEnquiryResponse* message exchanged between the DS and OS.

Due to similarity of the configuration parameters for our homecare application types, we have used the same data model for all the application types (e.g., blood pressure, weight and medication monitoring). The data model consist of several parameters which are explained as follows:

– *start_process*: it is a *boolean* variable to indicate whether it is needed to start the process (orchestration) of a homecare application. For instance, if the blood pressure is measured within the last two hours and there is no need for the alert, this variable is false and the process of MdM application will not start.

– *send_reminder*: it is a *boolean* variable to indicate whether it is needed to send a reminder for instance, if the medication is taken on time, this variable is false and no reminder message will be sent.

– *wait_before_first_reminder*: it is a *integer* variable to indicate that how many minutes the application should wait before sending the first reminder. The maximum value can be 30 minutes. Thus, all the orchestration instances are triggered by the DS half an hour before the scheduled time. Then, if this variable is set to 10, it means that the application sends the first reminder 20 minutes before the scheduled time.

– *reminder_repetition*: it is a *integer* variable to indicate the maximum number of sending reminder messages.

– *wait_before_reminder_repetition*: it is a *integer* variable to indicate that how many minutes the application should wait before sending the next (e.g., second or third) reminder.

– *reminder_message*: it is a *string* variable to set the reminder message for instance, "*Jan, please go home and take your medication*".

- *reminder_ modality*: it is an *enumerated* variable to set the modality of the reminder message for instance, "*audio*" and "*text*".

- *reminder_ binding_ port*: It is a *string* variable to set the binding address of the reminder application service for instance, " http://localhost:8080/Ucare-Applications/reminder?wsdl". As such, we assume that the DS knows the available application services at runtime.

- *send_ alert*: it is a *boolean* variable to indicate whether it is needed to send an alert for instance, if measured blood pressure value is not in the normal range.

- *alert_ message*: it is a *string* variable to set the alert message for instance, "*Jan's blood pressure is 12/18, go to his apartment immediately*".

- *alert_ interface*: it is an *enumerated* variable to set the interface of the alert message for instance, "*SMS*" and "*GTalk*".

- *alert_ binding_ port*: It is a *string* variable to set the binding address of the alert application service.

- *call_ medicine_ dispenser*: it is a *boolean* variable to indicate whether it is needed to call the MdM application from the VsM application. For instance, if the systolic of Jan's blood pressure is higher than 14 but less than 18, he should take a medication and then this variable becomes true.

- *care_ giver_ id*: It is a *string* variable to assign a care-giver to an application to receive the alert messages.

- *orchestration_ instance_ id*: it is a *integer* variable which is created when an orchestration pattern is triggered by the DS. Then in all the message exchanges between the DS and the OS, this is used as a unique reference id.

- *service_ plan_ id*: it is an *integer* variable which is created when a care-giver creates an application for a care-receiver. All the configuration parameters are stored in the data base with this unique id. The data base schema will be explained in Section 8.3.

The data model is also used for the notification message which is sent by the DS to notify the OS. The values of the variables of a data model can be *null* or not used by the OS if it is not required within a specific interaction. For instance, the *reminder repetition* element is null when there is no need to send a reminder.

In our application scenarios, the data model instances were less than 1 KB and therefore the whole data model is exchanged in each transaction. However, if there are many variables, the data model can be divided into a set of modules in order to reduce the amount of exchanged data between the DS an OS. We discuss this idea as our future work in Chapter 10.

8.2 Application Services

In this section, we explain how the application services were implemented on top of the application server (shown in Fig. *8-1*). The application services are the implementation of the SBBs which we explained in Section 5.1.1. As such, an application service implements the functionalities of its corresponding SBB. For instance, the blood pressure monitoring application service provided by MobiHealth implements the *Notify blood pressure()* functionality of *Blood pressure measurement* SBB.

A SBB might be implemented by two application services, one on the service provider platform and the another one on the DHSP platform. The location of an application service is determined based on who is calling that application service. For instance, the *calendar* SBB is implemented by two application services: (1) an application service, hosted on the BSS server, to implement the *Set agenda()* functionality which is called by the DHSP platform and (2) an application service, hosted on the DHSP platform, to implement the *Notify calendar event()* functionality which is called by the BSS server.

The configuration parameters of a SBB are used by the DS in order to adapt the application behaviour, for instance all the configuration parameters of the *Blood pressure measurement* SBB. Nevertheless, some of the configuration parameters might be passed to the application service as its input message for instance, *message* parameter of the *reminder* SBB is also given to the reminder application service to define the textual reminder message.

Each application service provider has the freedom to implement its related SBBs with respect to its requirements. The service provider can use its own message formats and deployment architecture. Therefore, we explain the application services based on their providers in the

following subsections.

8.2.1 Application Services Provided by MobiHealth

MobiHealth [106] provides three application services which implements the *Blood pressure measurement* , *Weight measurement* and *Oxygen saturation measurement* SBBs (see Section 5.1.1). The required vital-sign measurement device at home is connected to a server in MobiHealth company. The application services are hosted on our application server and acts as a data relay. Thus, they forward any vital-sign measurement values to the CS. To receive these values, first the application services should subscribe to the MobiHealth server. Listing 8.4 shows the WSDL specification of all the three vital-sign measurement application services.

```
<wsdl:types>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="http://mobihealth.com/datarelay/schemas">
<xs:element name="MethodResponse" type="tns:MethodResponse"/>
<xs:element name="NotifyBPRequest" type="tns:NotifyBPRequest"/>
<xs:element name="NotifySatRequest" type="tns:NotifySatRequest"/>
<xs:element name="NotifyWeightRequest" type="tns:NotifyWeightRequest"/>

<xs:complexType name="NotifyBPRequest">
  <xs:sequence>
    <xs:element form="qualified" name="Diastolic" type="xs:int"/>
    <xs:element form="qualified" name="PatientId" type="xs:string"/>
    <xs:element form="qualified" name="PulseRate" type="xs:int"/>
    <xs:element form="qualified" name="Systolic" type="xs:int"/>
    <xs:element form="qualified" name="TimeStamp" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="NotifySatRequest">
  <xs:sequence>
    <xs:element form="qualified" name="PatientId" type="xs:string"/>
    <xs:element form="qualified" name="Saturation" type="xs:int"/>
    <xs:element form="qualified" name="TimeStamp" type="xs:string"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="NotifyWeightRequest">
 <xs:sequence>
  <xs:element form="qualified" name="PatientId" type="xs:string"/>
  <xs:element form="qualified" name="TimeStamp" type="xs:string"/>
  <xs:element form="qualified" name="Weight" type="xs:int"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="MethodResponse">
 <xs:sequence>
  <xs:element form="qualified" name="ResultCode" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
</wsdl:types>

<wsdl:message name="MethodResponse">
  <wsdl:part element="sch:MethodResponse" name="MethodResponse"></wsdl:part>
</wsdl:message>

<wsdl:message name="NotifyBPRequest">
<wsdl:part element="sch:NotifyBPRequest" name="NotifyBPRequest"></wsdl:part>
</wsdl:message>

<wsdl:message name="NotifySatRequest">
<wsdl:part element="sch:NotifySatRequest" name="NotifySatRequest"></wsdl:part>
</wsdl:message>


<wsdl:message name="NotifyWeightRequest">
<wsdl:part element="sch:NotifyWeightRequest" name="NotifyWeightRequest">
</wsdl:part>
</wsdl:message>
```

```
<wsdl:portType name="DataRelay">

<wsdl:operation name="NotifyBP">
 <wsdl:input message="tns:NotifyBPRequest" name="NotifyBPRequest"> </wsdl:input>
 <wsdl:output message="tns:MethodResponse" name="MethodResponse"></wsdl:output>
</wsdl:operation>

<wsdl:operation name="NotifySat">
  <wsdl:input message="tns:NotifySatRequest" name="NotifySatRequest"></wsdl:input>
  <wsdl:output message="tns:MethodResponse" name="MethodResponse"></wsdl:output>
</wsdl:operation>


<wsdl:operation name="NotifyWeight">
 <wsdl:input message="tns:NotifyWeightRequest" name="NotifyWeightRequest"></wsdl:
 input>
 <wsdl:output message="tns:MethodResponse" name="MethodResponse">
</wsdl:output>
</wsdl:operation>

</wsdl:portType>
```

Listing 8.4 : The specification of the three vital-sign measurement application services provided by MobiHealth

The WSDL shows three *complexType*s for the three vital-sign types: (1) *NotifyBPRequest* for blood pressure, (2) *NotifySatRequest* for oxygen saturation, and (3) *NotifyWeightRequest* for weight. Later, all these vital-sign types are converted to a uniform vital-sign types (shown in Listing 8.2) used by the infrastructure services.

The WSDL defines a port type with three operations : (1) *NotifyBP* to notify the latest blood pressure value, (2) *NotifySat* to notify the latest measured oxygen saturation, and (3) *NotifyWeight* to notify the latest measured weight. These three operations implement the functionalities of the three vital-sign measurement SBBs.

8.2.2 Application Services Provided by Innospense

Innospense [78] provides an automatic medicine dispenser which is accessible as an application service. This application service implements the *medicine dispenser* SBB (see Section 5.1.1). Since Innospense used a web portal application, it can provide a request-response functionality and thus implements the *get medication intake()* functionality of the *medicine dispenser* SBB. This application service is hosted on the web portal located within Innospense company. On the application server, there is a web service client to interact with this web service. Furthermore, this web service client also implements *set schedule()* functionality of the *medicine dispenser* SBB to enable the DHSP platform to define medication schedule on the dispenser device. Listing 8.5 shows the WSDL specification of the automatic medicine dispenser application service. We only show part of its specification which is related to the two aforementioned functionalities of the *medicine dispenser* SBB.

```
<xsd:complexType name="scheduleArray">
 <xsd:all>
```

```
  <xsd:element name="id" type="xsd:int"/>
  <xsd:element name="client_id" type="xsd:int"/>
  <xsd:element name="date_time" type="xsd:string"/>
  <xsd:element name="amount" type="xsd:string"/>
  <xsd:element name="dispensed" type="xsd:int"/>
  <xsd:element name="forgotten" type="xsd:int"/>
  <xsd:element name="time_taken" type="xsd:string"/>
  <xsd:element name="custom_alarm" type="xsd:string"/>
 </xsd:all>
</xsd:complexType>

<xsd:complexType name="databaseReturn">
 <xsd:all>
  <xsd:element name="result" type="xsd:int"/>
  <xsd:element name="message" type="xsd:string"/>
 </xsd:all>
</xsd:complexType>

<xsd:complexType name="scheduleArrayList">
<xsd:complexContent>
  <xsd:restriction base="SOAP-ENC:Array">
  <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:scheduleReadArray[]"/>
  </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="scheduleArrayListReturn">
 <xsd:all>
  <xsd:element name="list" type="tns:scheduleArrayList"/>
 </xsd:all>
</xsd:complexType>
...
<message name="getPeriodRequest">
 <part name="soap_username" type="xsd:string"/>
 <part name="soap_password" type="xsd:string"/>
 <part name="client_id" type="xsd:int"/>
 <part name="date" type="xsd:string"/>
 <part name="days" type="xsd:string"/>
</message>

<message name="getPeriodResponse">
<part name="return" type="tns:scheduleArrayListReturn"/>
</message>

<message name="addRequest">
  <part name="soap_username" type="xsd:string"/>
  <part name="soap_password" type="xsd:string"/>
  <part name="changes" type="tns:scheduleWriteArray"/>
  </message>

<message name="addResponse">
  <part name="return" type="tns:databaseReturn"/>
</message>
...
<operation name="getPeriod">
<documentation>retrieves all the entrie(s) from the schedule for the given period.</
documentation>
<input message="tns:getPeriodRequest"/>
<output message="tns:getPeriodResponse"/>
</operation>

<operation name="addSchedule">
<documentation>Adds the specified entry from the schedule.</documentation>
<input message="tns:addRequest"/>
<output message="tns:addResponse"/>
</operation>
```

Listing 8.5 : Part of the specification of the automatic medicine dispenser application service provided by Innospense

Innospense defines *scheduleArray* as a *complexType* either to send or to receive medication schedules. Each medication schedule has a unique *id* which is correspondent to an event id defined on the DHSP platform. To implement *set schedule()* functionality, the DHSP platform sends a list of *scheduleArray* (*scheduleArrayList*) through the *addSchedule* operation. Innospense application service sends an acknowledge with sending back the *databaseReturn* message. This list are build based on the calendar event which are deployed by the tailoring platform.

To implement *get medication intake()* functionality, the DHSP platform call the *getPeriod* operation to take a list of *scheduleArray*, for a specific period of time. The *addRequest* message has a user-name and password to authenticate for each transaction, and *date* and *days* to specify a specific period of time. Innospense application service sends back the *scheduleArrayListReturn* as a list of *scheduleArray*. The *dispensed*, *forgotten* and *time_ taken* elements of the *scheduleArray* convey the medication intake information. The DHSP platform store the information in its data base which will be explained in Section 8.3.

8.2.3 Application Services Provided by BSS

BSS [140] provides calendar, reminder, (vital-sign or medication records) reporting and manual medicine dispenser application services. These application services implement thier corresponding SBBs (see Section 5.1.1).

BSS provides some of the functionalities of these SBBs, which must be called by the DHSP platform, as different operations of one web service so-called *WebAPI*. The *WebAPI* is hosted on an application server of the BSS and provides 26 operations to add, edit, delete and query the data used by the provided application services. Listing 8.6 shows part of the WSDL specification of the *WebAPI*.

```
...
<xs:complexType name="addAgendaItem">
 <xs:sequence>
 <xs:element name="event_id" type="xs:string" />
<xs:element name="care_receiver_id" type="xs:int"/>
 <xs:element name="startDate" type="xs:string" />
<xs:element name="endDate" type="xs:string" />
<xs:element name="notificationTimeBefore" type="xs:string" />
<xs:element name="location" type="xs:string" minOccurs="0"/>
<xs:element name="weblink" type="xs:string" minOccurs="0"/>
<xs:element name="email" type="xs:string" minOccurs="0"/>
<xs:element name="commentary" type="xs:string" minOccurs="0"/>
<xs:element name="category" type="xs:string" minOccurs="0"/>
<xs:element name="pictogram_url"
type="xs:string" minOccurs="0"/>
</xs:sequence>

<xs:complexType name="addAgendaItemResponse">
 <xs:sequence>
  <xs:element name="return" type="xs:int"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="sendReminder">
 <xs:sequence>
  <xs:element name="subject" type="xs:string" />
  <xs:element name="message_text" type="xs:string" />
  <xs:element name="modality" type="xs:string" />
  <xs:element name="care_receiver_id" type="xs:int"/>
  <xs:element name="event_id" type="xs:int"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="sendReminderResponse">
 <xs:sequence>
 <xs:element name="return" type="xs:int"/>
 </xs:sequence>
</xs:complexType>
...
<message name="addAgendaItem">
  <part name="parameters" element="tns:addAgendaItem"/>
  </message>

<message name="addAgendaItemResponse">
```

```
  <part name="parameters" element="tns:addAgendaItemResponse"/>
</message>

<message name="sendReminder">
 <part name="parameters" element="tns:sendReminder"/>
</message>

<message name="sendReminderResponse">
 <part name="parameters" element="tns:sendReminderResponse"/>
</message>
...
<operation name="sendReminder">
 <input wsam:Action="http://ws/WebAPI/sendReminderRequest" message="tns:sendReminder
 "/>
 <output wsam:Action="http://ws/WebAPI/sendReminderResponse" message="tns:
 sendReminderResponse"/>
</operation>

<operation name="addAgendaItem">
 <input wsam:Action="http://ws/WebAPI/addAgendaItemRequest" message="tns:
 addAgendaItem"/>
 <output wsam:Action="http://ws/WebAPI/addAgendaItemResponse" message="tns:
 addAgendaItemResponse"/>
</operation>
...
```

Listing 8.6 : Part of the specification of the *WebAPI* provided by BSS

BSS defines *addAgendaItem* as a *complexType* to send or to receive calendar events. The event id for each calendar event is unique and used by the BSS application server to notify the DHSP platform. Using the *startDate* and *endDate*, the DHSP platform deployed the time of calendar event during the deployment. The *notificationTimeBefore* is used by the BSS server, to know how many minutes in advance, it should notify a calendar event. There are several data type conversions which are done by the adapters hosted on the DHSP platform. For instance, the DHSP platform uses *integer* (e.g., 30 minutes) for *notificationTimeBefore* and it must be converted to *string* in order to deploy into the BSS calendar service.

A calendar event deployment is done through the *addAgendaItem* operation and in return, the BSS sends back an acknowledge indicating that the event is successfully deployed (*addAgendaItemResponse*). We have defined several integer response codes which are shared by all the application providers for instance, *addAgendaItemResponse*=0 means that the deployment is successfully done.

In addition, the *WebAPI* implements the *send message()* functionality of the *reminder* SBB using the *sendReminder* operation. The *sendReminder*, as an input data type, indicates the message subject, text and modality of the reminder message. In returns, the BSS acknowledge sending the reminder by sending back the *sendReminderResponse* message.

The *Notify calendar event()* functionality of the *calendar* SBB is implemented by another application service which is hosted on the DHSP platform. Listing 8.6 shows the specification of this application service.

```
<xs:complexType name="UcareCalendarNotify">
 <xs:sequence>
 <xs:element name="cr_id" type="xs:string" minOccurs="0"/>
 <xs:element name="event_id" type="xs:int" />
 <xs:element name="event_type" type="xs:string" minOccurs="0"/>
 <xs:element name="start_date" type="xs:string" minOccurs="0"/>
 <xs:element name="end_date" type="xs:string" minOccurs="0"/>
 <xs:element name="reminder_time_before" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="UcareCalendarNotifyResponse">
 <xs:sequence>
  <xs:element name="return" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<message name="UcareCalendarNotifyRequest">
 <part name="parameters" element="tns:UcareCalendarNotify"/>
</message>

<message name="UcareCalendarNotifyResponse">
 <part name="parameters" element="tns:UcareCalendarNotifyResponse"/>
</message>

<operation name="UcareCalendarNotify">
 <input wsam:Action="http://adapter.prov.ucare/calendar/UcareCalendarNotifyRequest"
 message="tns:UcareCalendarNotify"/>
 <output wsam:Action="http://adapter.prov.ucare/calendar/UcareCalendarNotifyResponse"
 message="tns:UcareCalendarNotifyResponse"/>
</operation>
```

Listing 8.7: The specification of the *calendar* application service (The DHSP platform side) provided by the BSS

The *UcareCalendarNotify* operation is called by the BSS in *notificationTimeBefore* in advance to notify a calendar event. The input message, *UcareCalendarNotifyRequest*, has the event id by which the DHSP platform can retrieve the corresponding data from its data base. The DHSP platform sends an acknowledge to indicate that the calendar event is received using the *UcareCalendarNotifyResponse*.

The *Notify medication intake* functionality of the *medicine dispenser* SBB is implemented by an application service which is hosted on (the application server of) the DHSP platform. Listing 8.8 shows the specification of this application service.

```
<xs:element name="confirm_medication" type="tns:confirm_medication"/>
<xs:element name="confirm_medicationResponse" type="tns:confirm_medicationResponse"/>

<xs:complexType name="confirm_medication">
<xs:sequence>
  <xs:element name="event_id" type="xs:int"/>
  <xs:element name="status" type="xs:string" />
  <xs:element name="taken_time" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="confirm  medicationResponse">
 <xs:sequence>
  <xs:element name="return" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<message name="confirm_medicationRequest">
 <part name="parameters" element="tns:confirm_medication"/>
 </message>

<message name="confirm_medicationResponse">
 <part name="parameters" element="tns:confirm_medicationResponse"/>
</message>

<operation name="confirm_medication">
```

```
  <input wsam:Action="http://adapter.prov.ucare/medicine_dispenser/
  confirm_medicationRequest" message="tns:confirm_medication"/>
 <output wsam:Action="http://adapter.prov.ucare/medicine_dispenser/
 confirm_medicationResponse" message="tns:confirm_medicationResponse"/>
</operation>
```

Listing 8.8 : The specification of the manual medicine dispenser application service provided by BSS

BSS provides a web-based interface to press a button to confirm taking a medication. Then the BSS server calls the *confirm_ medication* operation of the medicine dispenser application service hosted on the DHSP platform. The BSS sends *confirm_ medicationRequest* message which includes the event id and status of the medication intake (e.g., "*taken*" and "*not-taken*"). In case the medication is taken, the timestamp of medication intake is also sent by the *taken_ time* element.

8.2.4 Internal Application Service

As an internal application service, the DHSP platform provides an alert application service to implement the *alert* SBB. This application service is hosted on (the application server of) the DHSP platform. Listing 8.9 shows the specification of the alert application service.

```
<xs:complexType name="sendAlert">
 <xs:sequence>
  <xs:element name="alert_message" type="xs:string" minOccurs="0"/>
  <xs:element name="alert_interface" type="xs:string" minOccurs="0"/>
  <xs:element name="cg_contact" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="sendAlertResponse">
 <xs:sequence>
  <xs:element name="return" type="xs:int" />
 </xs:sequence>
</xs:complexType>

<message name="sendAlert">
 <part name="parameters" element="tns:sendAlert"/>
</message>

<message name="sendAlertResponse">
 <part name="parameters" element="tns:sendAlertResponse"/>
</message>

<operation name="sendAlert">
<input wsam:Action="http://adapter.prov.ucare/alert/sendAlertRequest" message="tns:
sendAlert"/>
<output wsam:Action="http://adapter.prov.ucare/alert/sendAlertResponse" message="tns:
sendAlertResponse"/>
</operation>
```

Listing 8.9 : The specification of the manual medicine dispenser application service provided by BSS

It implements the *send alert()* functionality of the *alert* SBB using the *sendAlert* operation. The input data type , *sendAlert*, which includes the alert message, interface and care-giver id (to whom the alert must be sent). In our implementation, we only have one alert interface which is "*Google_ talk*".

As such the alert application service, first retrieves the Google id of the corresponding care-giver and second sends the alert message to

her Google talk application which is running on her PDA. In return, after getting message delivered confirmation from Google server, the alert application service sends back an acknowledge using the $sendAlertResponse$ message.

8.3 Database Structure

We classify the database structure, used by the DHSP platform, into three substructures. These three substructures are designed and used for different purposes individually. The whole database structure were implemented in DB2 [73]. In the following we explain the conceptual models of the three database substructures.

8.3.1 Database Structure for Service Plan

This part of the database structure is used to store a deployed service plan and its configuration parameters. Since, the service plan is also used by the tailoring platform, we designed the required database structure for the service plan together with the programmer [168]. Nevertheless, to have the complete explanation of our design, we explain the database structure of the service plan with respect to the terminology of this thesis in this section. Fig. *8-6* shows the conceptual model of the database structure which is used to store a deployed service plan.

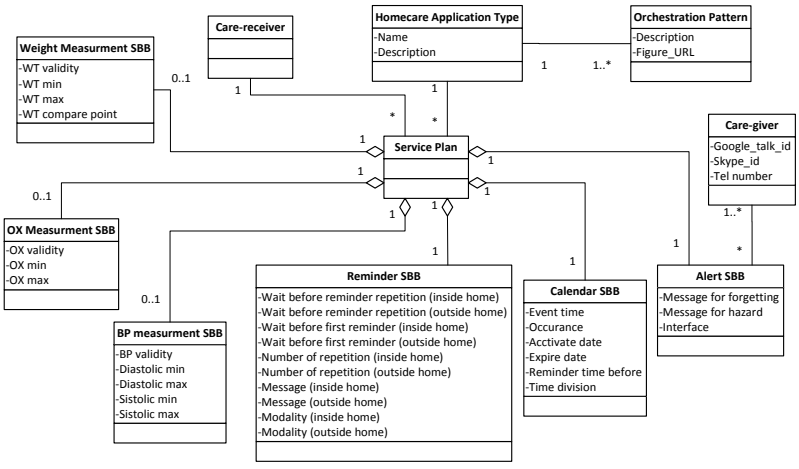

Figure 8-6 : The conceptual model of the database structure used to store a deployed service plan

A care-giver through the tailoring platform creates a service plan

for a homecare application type (the VsM, MdM or SaM application). Based on our design, we have only one service plan per pair of application type and care-receiver for instance, we have one blood pressure monitoring application (VsM) service plan for Jan as a care-receiver. When a new service plan is added, the DHSP platform deactivates the previous service plan by setting a flag in all the tables that have a record related to that service plan. Therefore, we can keep the history of service plan creations.

As discussed before, the service plan consist of two parts: configuration parameters and an orchestration pattern. The configuration parameters are stored in the corresponding tables of the 6 SBBs (e.g., *reminder SBB* table). The orchestration patterns are stored in the *orchestration patterns* and assigned to an application type. Then when a service plan is created for a specific application type, it only can use one of the orchestration patterns of that application type.

In addition, we have two other tables: *care-receiver* and *care-giver* tables. Despite the tailoring platform, the DHSP platform only keeps a hashed ids of the care-receiver (not the real name or id) in the *care-receiver* table. *Care-giver* table is used to store the contact information of the care-givers who receive alert messages.

During a service plan deployment, a new record is added to the it service plan table with a unique service plan id. In addition, based on the SBBs used by that service plan, new records are also added to the corresponding SBB tables. The tailoring platform gives all the required values of the configuration parameters through the *deployment server* explained in Section 5.2.2. We implemented a web service which can be called by the tailoring platform in order to send these values.

8.3.2 Database Structure for Provisioning Records

During the provisioning, the DHSP platform receives several data records from the application services which must be stored. These data can be used later by the *reporting* application service. To store these data records, the DHSP platform needs several database tables. Fig. *8-6* shows the conceptual model of the database structure which is used to store the data records during the provisioning of homecare applications.

Based on the *calendar SBB* table, the DHSP platform creates several calendar events and stores them in the *calendar event* table. These events are defined for a service plan and several service plans can be assigned to a care-receiver.

The calendar events could be either *social event*, *vital-sign event* or *medication event*. For each of these event types, the DHPS platform deploys the calendar event on the corresponding third-party service and stores the returned event id. For instance, the DHSP platform deploys

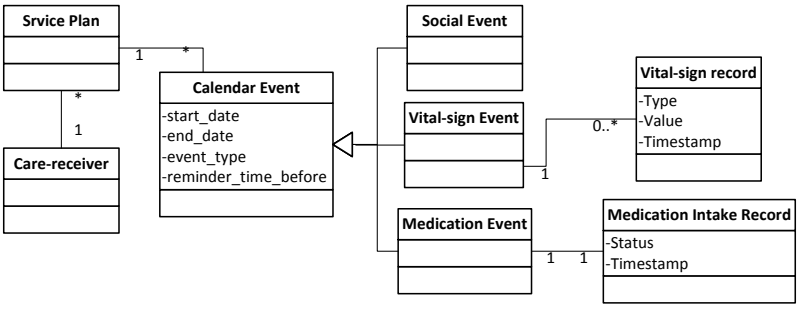

Figure 8-7 : The conceptual model of the database structure used by the provisioning data records

several calendar events of a VsM application into the calendar service provided by BSS and stores the returned event ids in the *vital-sign event* table.

One or several vital-sign measurement values which are stored in the *vital-sign record* table can be related to one vital-sign calendar. It enables the care-receiver to measure their vital-signs one or several times for one scheduled measurement event in their calendar. However, it is also possible that the care-receiver measured his vital-sign on his own request and therefore there is no corresponding calendar event for that measurement value.

Unlike vital-sign measurement, a medication intake record which is stored in the *medication intake record* table can be related to only one medication calendar event. Because the care-receiver can not take medication on his own request and must follow a predefined medication intake schedule. For social events, the DHSP platform does not store any records and it only need the *social event* table to stored the deployed event.

8.3.3 Database Structure for Provisioning Logs

After our field test, we analyzed the system logs to investigate the *effectiveness* and *efficiency* of the DHSP platform objectively. As such, the DHSP platform must store the logs of transactions among the (application or infrastructure) services. and thus it needs a database structure to support. To store the transaction logs, the DHSP platform needs several database tables. Fig. *8-8* shows the conceptual model of the database structure which is used to store the transaction logs during the provisioning of the homecare applications.

We defined all the services in the *service* table. Each service is either an application service (e.g., alert service) or infrastructure service (e.g., orchestration service) and has a unique name. Each service has several

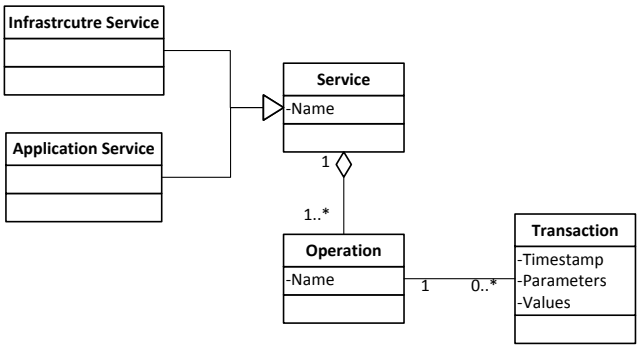

Figure 8-8 : The conceptual model of the database structure used by the transaction logs

operations which also have unique names (e.g., send_reminder and update_data_model). Several transaction logs can be defined for one operation which are stored in the *transaction* table. Each transaction log has the time stamp, sent or received parameters and their values. In our field test, we had 26 operations and in some of them we write the transaction logs twice, for instance, in the notify_blood_pressure operation, we wrote a transaction log when a vital-sign measurement is received and and another transaction log when the acknowledge is sent. This enabled us to measure the time spent by the DHSP platform to react to a new vital-sign measurement as a contextual change.

8.4 Deployment Architecture

As shown in Fig. *5-6*, the DHSP platform has three subsystems: (1) *infrastructure Server*, (2) *application Server*, and (3) *deployment Server*. Each of these subsystems has one or several components. In this section we explain how these components are deployed in our field test. Since we implemented all the application and infrastructure services as web services, they can be deployed on different services at different locations as long as these services can communicate with each other over the Internet for instance, through SOAP [150]. Therefore, we have several alternative deployment architectures and we chose the one which suits the requirement of our stakeholders the most. Fig. *8-9* depicts how the DHSP platform has been deployed.

– *Infrastructure server*: Infrastructure server as one of the three subsystems of the DHSP platform is running on a PC with Windows 2008 server operating system [163]. It has the three infrastructure component: *process engine*, *rule engine* and *context manger*

(see Section 5.2). As a process engine, we used WebSphere Lombardi Edition [74] and as a rule engine we used WebSphere ILOG JRules [75]. As discussed in Section 8.1, we implemented the context service as part of the decision service and thus the *context manger* was also running on the rule engine where the decision service was running.
In our field study, all the application services, were accessible by the infrastructure services through SOAP protocol, therefore, the infrastructure server can be located at our back office and this eases the maintenance of the server for us and the programmer.
The proposed rule engine has a GUI to edit the rules called rule editor which can only be installed on Windows platform. As such, we chose Windows 2008 server [163] for the infrastructure server.
On the two components of the infrastructure server (process and rule engine), the three infrastructure services (orchestration, decision and context services) were running as it is shown in Fig. *8-9*. All the application services communicate with these three infrastructure services through the SOAP protocol.

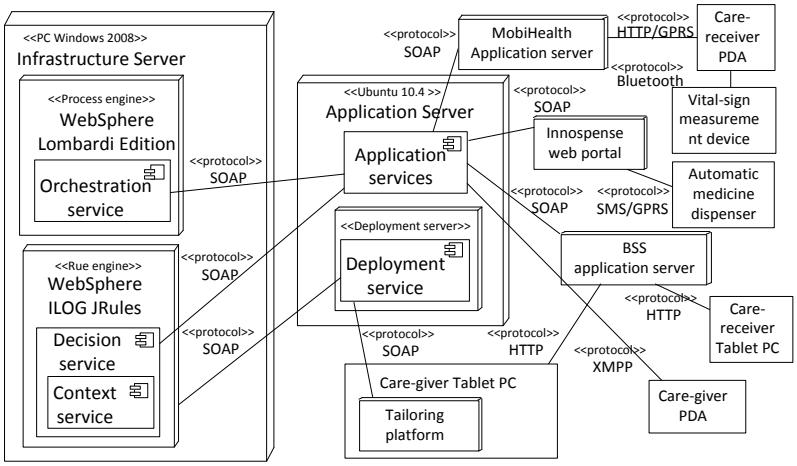

Figure 8-9 : The deployment architecture of the DHSP platform in our field test

– *Application server*: The application services on the application server must be able to communicate with their service providers' heterogeneous hardware and software components. In the homecare domain, some of the hardware components such as location sensors are resource-constrained and therefore, only capable to use network technologies with limited connectivity such as Bluetooth [159]. On the other hand, it is not financially feasible to put a server inside each care home. In our field test, all the service

providers' components are accessible trough the SOAP protocol and thus, although our application server can be located inside the care home, it is also located in the back office for the ease of maintenance.
We setup the application server on a Ubuntu (version 10.4) [143] machine. All the application services (and adapters) were implemented in an open source application server called GlassFish3 [62]. Its openness and support of all Java EE API specifications such as web services, XML and JMS (Java Message Service), seems promising in order to have heterogeneous types of adapters.
The vital-sign measurement application services communicate with a server located in Mobihealth [106] through SOAP protocol and this server communicates with a care-receiver's PDA through the GPRS protocol [160]. On top of the GPRS, the PDA uses HTTP [161] protocol to send the measured vital-sign values to the MobiHealth server. The PDA collects the vital-sign measurement values through Bluetooth [159] protocol and then forwards the values to the server of MobiHealth.
The medication monitoring application service communicate with the web portal of Innospense [78] through SOAP protocol and this web portal communicates with the medicine dispenser devices locate at the care homes through the GPRS protocol. On top of the GPRS, the medicine dispenser uses SMS [162] protocol to send the medication taken timestamps to the Innospense server.
The calendar, reminder, and other application services provided by BSS [140] communicates with the application server of BSS and the BSS server communicates with a web-based application running on the Tablet PCs of the care-giver and care-receivers through the HTTP protocol.
For the alert application service, we used a Google talk client that communicates with the PDA of the care-givers through the Extensible Messaging and Presence Protocol (XMPP) protocols. As such, a care-giver should sign in on a Google talk application on her PDA.

– *Deployment server*: The deployment server is implemented as part of the application server. It is implemented as a web service that communicates with the tailoring platform through the SOAP protocol. The tailoring platform is running as an application on the Tablet PC (or laptop) of a care-giver. When a care-giver confirms deployment of a service plan, the tailoring platform sends the service plan and the values of its configuration parameters to

the deployment service. Then the deployment service sends the deployed service plan to the decision service and context service (running on the rule engine) using the SOAP protocol. Since the required orchestration patterns are predefined on the orchestration service, the deployment service does not need to communicate with the orchestration service. All the decision rules (the configuration and composition rules) are predefined on the rule engine and during the deployment the rules are instantiated with the values of the configuration parameters sent by the tailoring platform.

Chapter 9

Validation

"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted."
— Albert Einstein

As discussed in the previous chapter, as a proof of concept, we have developed a prototype of the dynamic homecare service provisioning (DHSP) platform. The prototype is validated using a near real-life setting field test including two experiments. During the field test, the DHSP platform was used in daily use with more than 400,000 transactions among the infrastructure and application services. The goal of the field test was to study the usability of the DHSP platform in terms of (a) *effectiveness*, i.e., to see the number of useful system tasks such as sending an alert (adaptivity goals), deploying a service plan (tailorability goals) or modifying the application logic (evolvability goals), which have been completed within a specific time without systems errors, (b) *efficiency*, i.e., to see whether the completion time of the system tasks is acceptable by the stakeholder, and (c) *end-user satisfaction*, i.e., to see whether the perceived effectiveness and perceived efficiency of the system (consists of provisioning and tailoring platforms, and application services) from the end-users' point of view meets their expectations. For the satisfaction, we ask care-givers also about the care-receivers' opinion about using the U-Care system. Furthermore, we investigate the possible improvements based on the care-givers' feedback.

This chapter is organized as follows: Section 9.1 describes our validation criteria and evaluation strategy for evaluating our DHSP platform. Section 9.2 presents the setup of the field test and the role of involved organizations (including service providers and the care center). Section 9.3 presents the results of the first experiment, and how the system evolved based on the changes requested by the care-givers.

Section 9.4 discusses the results of the second experiment, how the system improved, and the lessons learned from the field test. Section 9.5 concludes our field test with an outlook on our future research.

9.1 Validation Criteria

We aim to provide a DHSP platform that can provide the following usability goals:

– To adapt a homecare application successfully within a certain time expected by an end-user

– To deploy a (re)tailored homecare application successfully within a certain time expected by a care-giver

– To deploy an evolved homecare application successfully within a certain time expected by a programmer

We have performed a field test to evaluate the usability of the DHSP platform. Usability is a multidimensional characteristic which must be measured in the context of users performing a task with a system in a specific environment [25]. Most usability evaluation methods gather both subjective and objective quantitative data. Subjective data are measures of participants' opinions or attitudes concerning their perception of usability. Objective data are measures of participants performance, such as deployment completion time and completion rate [132].

To evaluate the usability of our DHSP platform, we follow the ISO 9241-11 definition of usability, which is "*Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*" [80]. The standard provides guidelines to measure the effectiveness and efficiency, which results in objective data (through system transaction logs), and to measure satisfaction, which delivers subjective data (through end-users' interviews). Being able to combine objective and subjective measurements, would be useful to know which level of effectiveness and efficiency is acceptable in the homecare domain. Moreover, we identified explanations of our observations that allowed us to understand which parts of our approach need further improvement.

9.1.1 Effectiveness

The ISO standard defines effectiveness as "*accuracy and completeness with which users achieve specified goals*". In our case, we define it as the number of useful system tasks such as sending an alert (adaptivity goals), deploying a service plan (tailorability goals) or modifying the application logic (evolvability goals), which have been completed within a specific time without systems errors. Since homecare systems are real-time reactive systems [155], task completion is measured time-dependently. For instance, sending a late alert or sending a reminder when it is not needed, is considered an error, although the system task is completed. Moreover, we exclude the care-givers' mistakes. For instance, if a care-giver makes a mistake in tailoring an application but the deployment process goes well, we consider it as a completed task.

The effectiveness can be scored on a scale of 0 to 100%. In our field test, to measure the effectiveness, we will measure successful system task completion rate, i.e, the number of the system tasks that have been completed within a specific time as a percentage of the total number of system tasks.

$$system - task - completion - rate = \\ = \frac{completed - system - tasks * 100}{total - number - of - system - tasks}$$

9.1.2 Efficiency

The ISO standard defines efficiency as "*the level of effectiveness achieved to the expenditure of resources*". In our case, we define it as the completion time of the system tasks. There are other interpretations of efficiency such resource consumption and scalability. However, we have not considered them since we were not able to measure them in our field test.

To evaluate the completion time, we should compare it with other homecare systems. Since in our case there are no other systems to compare, we evaluate the completion times with the care-givers' expectations to see if the completion times are acceptable or not.

9.1.3 End-user satisfaction

The ISO standard defines satisfaction as "*the extent to which users are free from discomfort, and their attitudes towards the use of the product*". In our case, we define it as the perceived effectiveness and perceived efficiency of the system from the end-users' point of view. The satisfaction has some other aspects which are more related to interaction with the end-users [57], however, our platform interacts indirectly with the end-users through the application services or the

tailoring platform. Thus, to evaluate the satisfaction of the DHSP platform, we only consider the perceived effectiveness and efficiency. Moreover, we are interested to see how satisfaction can be affected by using different application services.

In order to measure satisfaction, we used questionnaires. Since we limit the satisfaction to the perceived effectiveness and efficiency, the existing usability questionnaires, that takes some other aspects into account [57], do not suit our requirements. Therefore, we have designed a questionnaire to ask specific questions about the perceived effectiveness and efficiency for each type of system task. Our questionnaire contains 8 open-end attitudinal questions to uncover end-users' beliefs and thoughts on the effectiveness and efficiency of each type of system task.

9.2 Setup of the Experiments

The field test has been done in two experiments of two months each, with one month in between to improve existing applications and to add new applications. The field test is an action case study [158], in which we aim to improve the current situation of providing care by using a DHSP platform. We follow the guidelines described by Wieringa in [157] to perform the experiments systematically.

Each series of the experiments was conducted in a near real-world setting in a care institute in the Netherlands and each series lasted for two months. The experiments are close to a real-world setting, because some real-world aspects are present, such as real care-receivers, real institution, real nurses and realistic scenarios, but other aspects are absent, such as only a single homecare institution, a few users and candies instead of real medicines. In this section, first, we explain the scenarios to be used in the experiments, then we describe which actors participate in the experiment and the role of each one (e.g., which service provider provides which application service) and finally, we explain the measurement tools and how we collect data.

9.2.1 Homecare Applications

In our field test, we have three types of homecare applications: 1) vital-sign monitoring (VsM), 2) medication monitoring (MdM) and 3) social activity monitoring (SaM). For VsM, we consider three types of vital-signs: blood pressure (BP), oxygen saturation (OX) and weight (WT). For MdM, we have two types of monitoring: using an automatic dispenser and using a manual dispenser.

Due to specific requirements of the care-receivers who participated

in the field test, we have used combinations of these applications. We involved the care-givers while defining these combinations to make them as realistic as possible.

9.2.2 Actors

Our field test has been done at Orbis [115], a care-institution in the Netherlands. This institution owns residential blocks where elderly can live and receive care services that are provided by professional care-givers. The aim of this institution is to provide round-the-clock services to their care-receivers and at the same time to enable them to live an independent life as much and as long as possible. The institute provides 8 care-receivers and 3 care-givers as end-users for our field test.

There are 3rd-party application services which are used by our homecare applications: calendar, reminder, vital-sign measurement, medicine dispenser and reporting service. The calendar, reminder and vital-sign reporting services are provided by the Biomedical Signals and Systems (BSS) group of the University of Twente [140]. These services are running on Tablet PCs available to the care-givers and care-receivers. The tailoring platform is provided by the Information System (IS) group of the University of Twente [141]. It is running as an application on Tablet PCs available to the care-givers. If the application logic needs to be updated manually to address unforeseen changes, a programmer of IS modifies the application logic and accordingly updates the service plan.

The vital-sign measurement services are provided by the MobiHealth [106] company. Care-receivers use a vital-sign measurement device, which is connected to a server in MobiHealth. The MobiHealth server forwards vital-sign measurement values (e.g., blood pressure), which it receives from the measurement devices, to the DHSP platform. We also have an alert service as an internal application of the provisioning platform. The alert service sends an alert to a care-giver's PDA when there is a hazard situation.

The automatic medicine dispenser is provided by the Innospense [78] company. Care-receivers use the automatic medicine dispenser, which is connected to a server in Innospense. The Innospense server forwards the medicine intake information (e.g., time stamps) to the DHSP platform. For the manual medicine dispenser, we use a simple box in combination with the reminder and alert services.

9.2.3 Measurement Instruments

To collect data, we logged all the interactions among the DHSP platform and application services. The system logs have the timestamp and the

data of the interactions. They are stored in a SQL database. At the end of each experiment, we analyzed them off-line.

To evaluate the evolvability, during the first experiment, we maintained a list of changes, which were requested by the care-givers. These changes had been applied after the first experiment. We investigated how much time was required to apply these changes. The changes can affect the DS (i.e., affecting the decision rules), the orchestration patterns or both. Generally speaking, updating the DS and its decision rules requires less time to be deployed into the DHSP platform than updating the orchestration patterns [171]. Later, in the second experiment, we investigated how effective the applied changes are.

After each series of the experiments, we had interviewed the care-givers and care-receivers who participated in the experiments using questionnaires. The first interview was about the perceived effectiveness and efficiency and the requested changes by the care-givers to improve the system for the second experiment. The second interview was about the perceived effectiveness and efficiency of the system in the second experiment similar to the first experiment (the last 9 questions) and also some general aspects such as pros and cons of using the proposed system (shown in Appendix *A-1*, *A-2*, and *A-3*).

9.3 The First Experiment and its Results

The applications used in the first experiment are: VsM (BP, WT, OX), MdM using a manual medicine dispenser and SaM. We explain the result of the first experiment according to the three types of dynamicity: $adaptivity$, $tailorability$ and $evolvability$, both objectively and subjectively.

9.3.1 Adaptivity

We group the system tasks into four categories: sending reminders, sending alerts, vital-signs measurements and medications dispensing. For reminder and alert, the functionality consists of sending the reminder or alert messages and receiving the acknowledge from the application service. For vital-sign and medication, the functionality consists of receiving the vital-sign or medication intake data, sending the data to the reporting service and sending back the acknowledge to the vital-sign measurement or medicine dispenser service. Table *9-1* shows how effective and efficient the application functionalities are adapted in the first experiment according to the system logs. For instance, during the first experiment (which lasted 2 months), with

8 care-receivers using the DHSP platform, the reminder task was in total 339 times executed, and 46 out of the 339 times the execution was not successful; and the duration of a single execution of this task was between 3419 ms and 10974 ms with an average of 6426 ms and standard deviation of 664 ms.

Table 9-1 : Effectivity and efficiency of adapting the functionalities of the applications (UnSucc = Unsuccessful, Rate = Success rate)

	Effectiveness			Efficiency (millisecond)			
	#Total	#UnSucc	#Rate	#Min	#Max	#Average	#Sdv
Reminder	339	46	%86.4	3419	10974	6426	664
Alert	171	16	%90.6	180	1379	356	176
Vital-sign	247	21	%91.4	6255	6712	6392	97
Medication	55	12	%78.1	165	11888	922	2217

For effectivity, the unsuccessful numbers of activities are calculated based on receiving exception errors or based on feedbacks from the care-givers. There are four reasons for the unsuccessful tasks: (1) operating system update: the DHSP platform was running on a Windows 2008 server. The operating system updates itself every day at 3 a.m. by default, (2) application update: during the experiment, one of the service providers updates its application services without informing the platform providers, (3) behavioral change of the care-receiver: after introducing the system, some care-receivers measure their vital-signs much earlier than expected, for instance at 5 a.m. instead of 8 a.m., which is the scheduled time. Then if a care-receiver's vital-sign values are too high or low, the application must send the alert immediately after 5 a.m. However, in the first experiment, the application starts only half an hour before the scheduled time to check the measured values. Thus, some alert messages were not sent on time, and (4) duplicated vital-sign measurement values: due to the Bluetooth network used by MobiHealth, sometimes a vital-sign value was sent twice to the platform and the platform forwards the value to the reporting service two times accordingly.

For efficiency, the system measures the time from sending (or receiving) data until receiving (or sending) an acknowledge. For instance, the time of vital-sign task is calculated from receiving a vital-sign value until sending back an acknowledge to the vital-sign measurement service. Based on the results from our first interview, we conclude that care-givers are not satisfied with the effectiveness of the system. Specially missing alerts are considered unacceptable in life-threatening situations. Besides, when care-receivers cannot use the system for several times, they are not interested in technical reasons and they may loose their trust and interest in using the system. However, the

efficiency was considered acceptable. The care-givers mention that the alerts must be delivered $immediately$ after the vital-signs measurements in case the values are higher or lower than a predefined threshold. We tried to quantify that and we found out that less than one minute would be considered as $immediately$ by the care-givers. Looking at the Table *9-1*, we can see that the efficiency of successful tasks are acceptable for the care-givers.

9.3.2 Tailorability

In the first experiment, care-givers can tailor or create three types of service plans: VsM, MdM (manual) and SaM. The tailorability task consists of receiving a new service plan by the DHSP platform, deploying the service plan to application services and sending back the acknowledge to the tailoring platform. This is done on the fly without interrupting running applications. Table *9-2* shows how effective and efficient is the tailorability of the applications. For instance, during the first experiment, VsM application was tailored in total 134 times, and 12 out of the 134 times the tailorability task was not successful; and the duration of a single execution of this tailorability task was between 296 ms and 85068 ms with an average of 5252 ms and standard deviation of 13202. There was only one reason for unsuccessful tailorability tasks: duplicated primary key: the tailoring platform enables care-givers to delete an existing service plan by deleting its events from the calendar service and adding a new service plan and its corresponding events. The calendar service used the Ids (i.e., identifier) of deleted events for the new events, while the tailoring platform used these Ids as primary key and does not delete them permanently to keep the activity logs of tailoring tasks. Therefore, for some tailorability tasks, the process was interrupted by duplicated primary key error.

Table 9-2 : Effectivity and efficiency of tailorability tasks (UnSucc = Unsuccessful, Rate = Success rate)

	effectiveness			Efficiency (millisecond)			
	#Total	#UnSucc	# Rate	#Min	#Max	#Average	#Sdv
VsM	134	12	%91	296	85068	5252	13202
MdD	30	5	%83.3	316	27337	3069	6458
SaM	270	15	%94.4	16425	97654	26780	19876

For efficiency, we measure the time from receiving a service plan until sending back the acknowledge to the tailoring platform. Based on the results from our first interview, we conclude that care-givers are satisfied with both effectiveness and efficiency of the tailorability. The number of unsuccessful tailorability tasks was tolerable by care-givers since there was no effect on the running applications.

9.3.3 Evolvability

During the first experiment and also our first interview, we collected the changes which are requested by care-givers (i.e.,unforeseen changes). Some of the changes are related to end-user interfaces of application services for instance, showing less text in the calendar service. Some other changes should be addressed on the DHSP platform and thus we investigate how the platform supports the evolvability of the applications. These changes are listed as follows: (1) If care-receivers measure their vital-signs earlier than the scheduled time and the measured values are not in the normal range, the alert must be sent immediately, (2) The vital-sign values should be sent with the alert messages to help care-givers to be prepared in advance, (3) For weight measurement, first we sent an alert when the weight of a care-receiver was either higher or lower than the last measured weight more than a threshold. Later on, care-givers want to receive this alert if the weight of a care-receiver is either higher or lower than a threshold without comparing with his last measured weight value, and (4) The nurses can add more than one event per day to the calendar service for each application.

The programmer modifies the application logic manually to evolve the applications based on the unforeseen changes. To address the unforeseen change 1, we add an alert activity at another orchestration pattern that receives vital-sign values from the MobiHealth. The alert activity calls the decision service of VsM application to see whether it is necessary to send alert or not. Since, we have reused the VsM decision service in another orchestration with the same service interface, the modification is accomplished quickly by dragging and dropping an alert activity to the orchestration. To address the unforeseen changes 2, 3 and 4, the modification is required only in the decision service without changing the orchestrations.

We improved evolvability of the applications because of using the decision service. However, using the decision service increased time and data communication to run the applications. In the first experiment, the decision service has been called 256 times. It takes 278 milliseconds by average (min=149,max=1700, Std=176 milliseconds) to call the decision service. This period of time is much less than the time durations of the system tasks (see Table *9-1*) and is not noticeable by care-receivers and care-givers. The data model of messages between the orchestration and the decision service consists of 19 variables (8 String, 7 Integer and 4 Boolean variables). In our field test, the size of data which is exchanged between an instance of orchestration and the decision service is always less than 5 kilobytes for each interaction. Therefore, we have seen that using of the decision service improves

evolvability with low cost in terms of its time and data communication.

9.4 The Second Experiment and its Results

Two months after the first experiment, the improved version of the system based on the requested changes was validated in the second experiment. The applications used in the second experiment are: VsM (BP, WT), MD (using manual and automatic dispenser) and SaM. In the second experiment, we empowered the system with an extra power supply, disabled automatic operating system update and asked all the service providers not to modify their application services during the experiment. We first evaluate the adaptivity and tailorability of the system to see if the system is improved. After execution of the second experiment, we interviewed the care-givers to measure their perceived *effectiveness and efficiency*. Moreover, we asked the care-givers to give us their opinions about the whole system through open-end (descriptive) questions. Even though the focus of this work is the DHSP platform and its implemented prototype, some of the results reported in this section are in general about the whole system.

9.4.1 Adaptivity

Based on the result, the effectiveness was improved since we had only four unsuccessful vital-sign tasks. The reason was that the reporting service took longer than the first experiment. Therefore, the DHSP platform sent an acknowledge to the vital-sign measurement service with a delay and it caused an error on care-receivers' PDAs. Although the vital-sign values are received and shown correctly, we consider them as unsuccessful task since showing this error on the PDA was inconvenient for care-receivers. The achieved efficiency of the second experiment is almost the same as the one in the first experiment. Based on our second interview, the care-givers are satisfied with both effectiveness and efficiency of the system tasks' adaptivity.

9.4.2 Tailorability

We have achieved 100% effectiveness regarding VsM and MdM tailorability. However, we had several failure for SaM tailorability because care-givers create more than 5000 social events per each service plan deployment. Thus, the deployment took more than 3 minutes and they stopped using the application since they thought it was broken. Based on our second interview, care-givers are satisfied with the effectiveness and efficiency of the VsM and MdM tailorability,

but not with the SaM tailorability.

9.4.3 General feedback

We experienced that not all the care-givers could distinguish the different parts of the system. Therefore, our second interview contains 20 task-independent questions to ask care-givers' opinions regarding the whole system as an integrated application. These are the lessons learned:

- Care-givers believe that the proposed homecare system can work in practice particularly after the second experiment. Note that the system is not used occasionally, but in daily use with more than 400,000 transactions among the services. However, the end-user interfaces for care-receivers must be improved (e.g., showing less text and bigger icons).
- The system would be more useful for private apartments outside of the care-center because it can save lots of traveling time for care-givers. Inside the care-center, it is sometimes faster to reach the care-receiver instead of using the system.
- In close future, care-receivers with less health problems will be advised by the government to stay at home in order to reduce healthcare costs and thus, the system would be even more promising.
- The system reduces physical contact that can decrease care quality. In this case, a voice or video communication can be helpful.
- Although adding vital-signs values to alert messages was useful, it is not sufficient to decide what they should do before visiting care-receivers. So, a video communication can help care-givers to judge.
- It is highly desirable that care-receivers measure their vital-signs without waiting for a care-giver particularly when they want to leave their apartments. However, care-receivers would be obliged to get back home, if they were outside their apartments at the scheduled time. Therefore, it is useful if they can carry the measurement devices with themselves.
- The field test can be improved if it takes longer than 4 months with better target group of care-receivers (e.g., 70-80 years old).
- An automatic dispenser is more successful than the manual one because first, it has an embedded reminder beep and pressing a button on the dispenser device is easier than pressing a message box on Tablet PC, and second, it cuts the bag that contains the medicine.

9.5 Conclusion

In this work, we discussed a field test of our dynamic homecare service provisioning (DHSP) platform. We investigated whether the following goals can be supported by the platform: (a) To adapt a homecare application successfully within a certain time expected by an end-user, (b) To deploy a (re)tailored homecare application successfully within a certain time expected by a care-giver, and (c) To deploy an evolved homecare application successfully within a certain time expected by a programmer. At least one of the above goals should be achieved in order to justify using the proposed DHSP platform. With respect to the these goals, we investigated the usability, i.e., efficiency, effectiveness and satisfaction (perceived efficiency and effectiveness) of the homecare applications running on the platform.

We believe that dynamic homecare applications provisioning must be efficient for the platform, care-giver, and programmer, i.e., the required time for adapting, (re)tailoring and evolving the applications with respect to the contextual changes must be as less as possible. Our interviews indicated that a desirable property of the DHSP platform is to decrease the work load of the care-givers, thus saving costs for the care centers and care-provider organizations. In addition, saving cost on manpower (needed for manual tasks) should not be annihilated by extra costs for system resources and application programmers. This cost saving for addressing new contextual changes is very important in the homecare domain since its environment is often subject to several types of changes. Our field test shows that the efficiency of adaptivity is acceptable for the care-giver although the response time is higher than that of a stand-alone application due to distributed application services. The efficiency of tailorability is also acceptable for the care-givers except for the SaM application due to the large numbers deployed events in each tailoring task. Separating the decision making rules from the orchestrations improves the evolvability of the applications since almost all the changes have been handled within the decision rules without redeploying the orchestration patterns.

Regarding the effectiveness, the first experiment was not successful due to several system failures. These failures caused several safety and availability risks using our proposed platform. However, it motivated us to develop a risks identification method which we applied after the first experiment to prevent similar risks in the second experiment. As a result, the effectiveness of the homecare applications in the second experiment is acceptable for the care-givers.

The care-givers believe that having a DHSP platform enables them to use alternative application services (e.g., manual or automatic

medicine dispenser) for a specific homecare application (e.g., MdM application) without affecting how the care-givers tailor and how the care-receivers use that homecare application. This would be useful to decrease the required time from both care-givers and care-receivers to learn how to use a new homecare application. Moreover, using the DHSP platform enables the care-giver to create applications with more realistic and useful functionalities. For instance, in our field test, the care-givers created a new application to ask a care-receiver to take a medication if his blood pressure is in a specific range, which can be done by integrating a medicine dispenser and blood pressure measurement devices.

Our conclusion from the field test is that the DHSP platform is usable (by care-givers and care-receivers) at least in our field test. However, the number of homecare applications and end-users involved in our field test is limited. To be able to provide statistical analysis, we plan to provision more homecare applications with more care-givers and care-receivers as our future work. In addition, our field test shows that both care-givers and care-receiver are interested to see more often measured vital-sign values. But it is important (1) how to show this data to the end-users (e.g., using graphical interface or statistical analysis) and (2) to export the data automatically to other healthcare applications (e.g., hospital patient record). As such, using application services with data processing user-interfaces and integrating the vital-sign measurement and medication intake data with other existing healthcare information systems are considered in our future plan.

Chapter 10

Conclusions and Future Work

"Whenever there is any doubt, there is no doubt."
— Ronin (1998)

This thesis proposed an architectural support for dynamic service provisioning in the homecare domain by introducing a *Dynamic Homecare Service Provisioning (DHSP)* platform. The platform supports three types of *dynamicity*:

– *Adaptivity*: is used when a given requirement of a care-receiver needs to be satisfied, and there are several predefined responses to satisfy the requirements which depend on the care-receiver's context. Then system adapts the response to the situation.

– *Tailorability*: is used when the care-receiver has a new requirement, and the response to this requirement has already been implemented, or can be automatically generated based on high-level instructions from a care-giver.

– *Evolvability*: is used when the care-receiver has a new requirement which is not foreseen at design time and thus no suitable response has been implemented. Therefore, at a later phase in the lifecycle, a programmer of a application must modify the application logic to deal with the new requirement.

The aim of our research was to provide the infrastructure for all three forms of dynamicity in a composite service-oriented application in the homecare domain. In addition, the platform integrates different IT-based services for the provisioning of homecare applications.

In the requirements elicitation phase of our approach, we interviewed professional care-givers in a care institute in the

Netherlands and also did a literature study of existing homecare systems. Moreover, we interviewed three homecare application providers to bring their requirements into account during the design our DHSP platform. Based on the identified requirements, we concluded that a hybrid service composition using a combination of process and rules is a promising approach for dynamic service provisioning in the homecare domain.

A prototype implementation of the proposed DHSP platform has been developed and evaluated in a near real-life setting. In this chapter, we discuss what we learned with respect to our research questions and objective, and then propose some challenges as potential future research topics.

This chapter is organized as follows: Section 10.1 summarizes the contributions of this thesis by answering the research questions and discusses to what extent, we have reached our research objectives. Section 10.2 discusses possible extensions to this work and identifies possible future research.

10.1 Contributions

The proposed DHSP platform in this thesis has the following usability goals:

- To adapt a homecare application successfully within a certain time expected by an end-user.
- To deploy a (re)tailored homecare application successfully within a certain time expected by a care-giver.
- To deploy an evolved homecare application successfully within a certain time expected by a programmer.

In the literature, usability is defined as the software quality attribute which comprises the efficiency, effectiveness and satisfaction aspects. We have performed a near real-life setting field test to evaluate the usability of the DHSP platform.

In Section 1.5, we presented the three top-level research questions and their corresponding sub-questions to be addressed in this thesis. In this section, we explain our contribution by reflecting on the results and describe how we addressed the research questions.

– **RQ1:** *Why do we need a DHSP platform?*

To answer this question, we needed to answer several sub-questions as follows:

1. **RQ1-1:** *What is a homecare service provisioning platform?*

   In Chapter 2, we defined a homecare service provisioning platform as part of a homecare system and its stakeholders such as care-givers and care-receivers. We explained how a homecare application is created by a care-giver and deployed to the provisioning platform through a tailoring platform. We also explained how the provisioning platform employs third-party and internal (application) services to execute a deployed service plan.

2. **RQ1-2:** *What are the requirements on this platform?*

   In Chapter 4, we identified functional and non-functional requirements that a homecare service provisioning platform should provide. The requirements were identified by interviewing of the stakeholders (mainly functional requirements) and literature study of existing homecare systems (mainly non-functional requirements).

3. **RQ1-3:** *What is dynamicity?*

   In Chapter 2 (Section 2.1), inspired by the identified requirements, we defined the three types of dynamicity which must be addressed in a dynamic service provisioning appraoch. As such, we explained different types of contextual changes (e.g., observable vs. non-observable) and which entities (an application itself, an end-user or a programmer) are responsible to monitor and then to address the changes.
   We also defined a dynamic service provisioning platform that supports applications for adaptivity, end-users for the tailorability of the applications, and programmers for the evolvability of the applications.

4. **RQ1-4:** *What is a homecare system?*

   In Chapter 2 (Section 2.4), we specialized the dynamic service provisioning platform in the homecare domain as a DHSP platform to address the dynamicity of the homecare domain.

In Chapter 3 (Section 3.1), we studied the exiting home solutions to investigate to what extent the dynamicity challenges have been addressed. To study the homecare solutions, we used our service provisioning framework (introduced in Section 2.5) to explain different challenges and functionalities of a homecare service provisioning platform.

Based on the identified requirements and existing homecare solutions, we concluded that there was a need for a DHSP platform, i.e., a homecare service provisioning platform that supports the dynamicity in the homecare domain. Therefore, we emphasized the dynamicity challenges and the design of a DHSP platform in our research explained in Section 3.1.2.

– **RQ2:** *How to design a DHSP platform?*

  We aimed to design a DHSP platform to improve the current homecare service provisioning situation with respect to the dynamicity challenges and the identified requirements. To design a DHSP platform, we needed to answer several sub-questions as follows:

1. **RQ2-1:** *What are the existing dynamic provisioning approaches?*

   In Chapter 3 (Section 3.2), we positioned our DHSP platform and its concepts with respect to related existing terminology and provides an overall view of existing dynamic service provisioning approaches. Then we studied the exiting hybrid service composition (processes and rules) approaches. With respect to the identified requirements, we concluded that a hybrid service composition using a combination of process and rules is a promising approach for dynamic service provisioning in the homecare domain.

2. **RQ2-2:** *What are the components of a DHSP platform?*

   In Chapter 5, we introduced our dynamic homecare service provisioning (DHSP) platform based on a hybrid service provisioning approach. We explained the SBBs, the types of decision rules, and how a service plan is defined and executed based on the SBBs and decision rules. We also introduced three infrastructure services to execute the deployed service plans.

3. **RQ2-3:** *How do these components interact with each other?*

   In Chapter 6, we zoomed into the three infrastructure services: process engine, rule engine and context manager, and explain how they interact with each other in more details. We separate decision-making rules (decision rules) from application process logic and then expose these rules as a *decision service*, which can be employed by the orchestration service to make adaptation decisions with respect to runtime contextual changes. As such, we introduced the decision service template including its service interfaces, their input and output messages, and their message exchange patterns to define the behaviour of the *decision service*.

4. **RQ2-4:** *How to identify the risks of using a DHSP platform?*

   The homecare applications are composed of application services provided by different, economically independent service providers. In the first experiment of our field test, we found that although the application services as actually delivered by the service providers meet their requirements, there is still a mismatch across service providers due to unstated assumptions. In Chapter 7, we introduced an Assumption-based Risk identification Method (ARM). The method identifies potential risks of using the DHSP platform due to mismatched unstated assumptions made by different service providers, and this mismatch causes an incorrect composite application to be delivered to end-users. The ARM method helped us to identify several risks before using the DHSP platform in the second experiment of our field test.

By answering all these four sub-questions, we had a complete design of our DHSP platform and then we explained its prototype implementation as a proof of concept in Chapter 8.

– **RQ1:** *What is the contribution of our DHSP platform?*

  The prototype of our DHSP platform was validated in a near real-life setting field test. In Chapter 9, we reported on two experiments which are performed in the field test and presented their results. During the field test, the DHSP platform was used in daily use with more than 400,000 transactions among the infrastructure and application services. The goal of the field test was to study the usability of the DHSP platform in terms of (a) *effectiveness*, (b)

efficiency, and (c) *satisfaction* both subjectively and objectively. We investigated the contribution of our proposed DHSP platform by observing how the homecare service provisioning situation is affected with respect to the three types of dynamicity as follows:

1. **RQ3-1:** *How to validate the adaptivity of the applications?* We observed that the adaptivity of the homecare applications met the end-users' (care-receiver and care-giver) expectations, at least in the second experiment of our field test.

2. **RQ3-2:** *How to validate the tailorability of the applications?* Except the social activity monitoring (SaM) application, we observed that the tailorability of the homecare applications met the care-givers' expectations, at least in the second experiment of our field test. For the SaM application, the tailorability was not acceptable by the care-givers due to huge number of events must be deployed per service plan and thus the long deployment time. The care-givers were satisfied with the fact that they only need to use the same tailoring application for all the homecare applications.

3. **RQ3-3:** *How to validate the evolvability of the applications?* We observed that the evolvability of the homecare applications met the programmers' expectations. This was possible mainly because of using the decision service. Most of the unforeseen changes (requested by the care-givers) were addressed within the decision rules without the need of changing and re-deploying the orchestration patterns.

10.2 Future Research

In this thesis, we have answered several research questions. While answering these questions, we lead to more research questions which should also be answered. Each of these new research questions could be a new research topic. In the following, we discuss five topics for future research that could improve the dynamic homecare service provisioning:

– **Other Field Tests**
We only evaluated the proposed DHSP platform in a care center in the Netherlands in which care-receivers live in their care homes.

Evaluation of the platform in other situations may have different results. For example, the DHSP platform should be tested in a situation where care-receivers live at their own homes and receive support from care- givers remotely or from family members. Care-receivers who live at home, might appreciate IT-based services and are more motivated in using them as opposed to care-receivers live in a nursing home. On the other hand, the usability of the homecare applications might be different (e.g., less efficiency because of more communication time) .
Moreover, the platform should be evaluated using other homecare application scenarios and theirs required SBBs and application services. For example, our field test shows that both care-givers and care-receiver are interested to see more often measured vital-sign values. However, it is important (1) how to show this data to the end-users (e.g., using graphical interface or statistical analysis) and (2) to export the data automatically to other healthcare applications (e.g., hospital patient records). As such, using application services with data processing user-interfaces and integrating the vital-sign measurement and medication intake data with other existing healthcare information systems could motivate the care centers to use the DHSP platform.

- **Other Types of Context**
  In our field test, we only used three types of context information namely: time, location and vital-sign values. We have also designed the DHSP platform and its decision rules only based on these three types of context information. We could consider other types of context information such as people nearby and care-receivers' current activity. Adding these types of context information could affect the decision rules and orchestration patterns. Currently we used a key-value ontology model to define the contextual events and IF-THEN-ELSE decision rules to adapt the application with respect to the contextual changes. However, this might not be efficient when we have several context types and thus we may need to use other context model and inference approaches such as Object Oriented Models [136].

- **Automatic Service Plan Creation**
  In our current design, a service plan supports a set of possible variations. A care-giver, as a domain expert, can select one of these variations by setting the values of the configuration parameters through a tailoring platform to address the contextual changes. If the implemented variations would not be sufficient, an

application programmer must manually create new variations. The DHSP platform can also support the care-givers to generate a new variation by providing a high-level non-technical domain-specific programming language. Then the care-giver can define the new situation (e.g., in natural language) and then the DHSP platform generates the service plan automatically.

– **Modularity of the Data Model and its Decision Rules**
In our application scenarios, the data model instances were less than 1 KB and therefore the whole data model is exchanged in each transaction. However, if there are many variables, the data model can be divided into a set of modules in order to reduce the amount of exchanged data between the infrastructure services. Therefore, based on the executed decision rules, the orchestration service only needs to update part of a data model instance which is changed. The modules of a data model are defined by its related decision rules. For instance, in the VsM application, all the reminder-related decision rules only could change the reminder-related values of the data model instance such as the message subject, text and modality of the reminder message.

– **Orchestration Patterns Adaptation**
In our current design, there are several predefined orchestration patterns and one of them can be selected based on the output of the decision rules at runtime. Thus the decision rules (and the decision service) can not manipulate the orchestrations patterns. However, our proposed approach can be extended by using worklets as a set of self-contained sub-processes [3, 4]. To do so, the output of decision rules should be defined based on worklets which can be added to an orchestration. For example, instead of having two orchestration patterns with and without an MdM call activity, we can have one orchestration pattern and one MdM call activity as a worklet. Then the decision rules determine which worklet and where within the orchestration pattern should be executed. With respect to exiting industrial process engines, supporting adding and removing worklets is considered as a challenge.

– **On-demand Service Provisioning**
We have observed during the field test that using service on demand, in which the care center can pay for the software and hardware infrastructure based on the number of its care-receivers, is highly desirable in the homecare domain. Cloud-computing and its pay-per-use basis capability could help the care centers to pay for

their homecare provisioning infrastructure based on the number of care-receivers. Specially using the decisions service enables the care centers to run computation-intensive decision-making rules on a cloud computing infrastructure. Furthermore, the care centers only have to dedicate cheap application services (and devices) with limited computational power to their care homes. However, several issues such as the trade-off between communication and computation cost should be investigated in order to justify on-demand service provisioning in the homecare domain.

References

[1] Health-EU. Elderly. `http://ec.europa.eu/health-eu/my_health/elderly/index_en.htm`. Accessed: 20/11/2012.

[2] *Pattern-oriented software architecture: On patterns and pattern languages*. 2007.

[3] Michael Adams, Arthur ter Hofstede, David Edmond, and Wil M.P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. 2006.

[4] Michael Adams, Arthur ter Hofstedee, Wil M.P. van der Aalst, and David Edmond. Dynamic, Extensible and Context-aware Exception Handling for Workflows. In *Int. Conf. on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS*, pages 95–112. Springer-Verlag, 2007.

[5] Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A service creation environment based on end to end composition of web services. In *Proceedings of the 14th international conference on World Wide Web*, pages 128–137. ACM, 2005.

[6] Marco Aiello and Schahram Dustdar. Are our homes ready for services? a domotic infrastructure based on the web service stack. *Pervasive and Mobile Computing*, 4(4):506–525, 2008.

[7] Alireza Zarghami and Brahmananda Sapkota and Mohammad Zarifi Eslami and Marten van Sinderen. Decision as a Service: Separating Decision-making from Application Process Logic. In *The 16h IEEE Int.l Enterprise Distributed Object Computing Conference (EDOC)*, pages 103–112. IEEE, 2012.

[8] Alireza Zarghami and Mohammad Zarifi Eslami and Brahmananda Sapkota and Marten van Sinderen. Dynamic Homecare Service

Provisioning Architecture. In *9th Int. Conf. on Service-Oriented Computing and Applications (SOCA)*, pages 292–299, 2011.

[9] Alireza Zarghami and Mohammad Zarifi Eslami and Brahmananda Sapkota and Marten van Sinderen. Service Realization and Compositions Issues in the Homecare Domain. In *6th International Conference on Software and Data Technologies (ICSOFT), Volume 1*, pages 347–356. SciTePress, July 2011.

[10] Alireza Zarghami and Mohammad Zarifi Eslami and Brahmananda Sapkota and Marten van Sinderen. Toward Dynamic Service Provisioning in the Homecare Domain. In *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth), Workshop on Designing and Integrating Independent Living Technology*, pages 292–299. IEEE, May 2011.

[11] Alireza Zarghami, Mohammad Zarifi Eslami, Marten van Sinderen, and Roel Wieringa. Dynamic homecare service provisioning: A field test and its results. In *Process Support and Knowledge Representation in Health Care*. Springer, June 2013.

[12] OSGi Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[13] João Paulo Almeida, Alberto Baravaglio, Mariano Belaunde, Paolo Falcarin, and Ernö Kovacs. Service creation in the spice service platform. In *17th Wireless World Research Forum Meeting (WWRF 17)*, pages 1–7, 2006.

[14] JA Alonso-Jimene, Joaquin Borrego-Diaz, Antonia M Chavez-Gonzalez, and Francisco J Martin-Mateos. Foundational challenges in automated semantic web data and ontology cleaning. *Intelligent Systems, IEEE*, 21(1):42–52, 2006.

[15] Amigo.
Ambient intelligence for the networked home environment, 2008. Available at: http://www.hitechprojects.com/euprojects/amigo.

[16] Apache. Apache ODE. Available at: http://ode.apache.org/, last visited: May 2013.

[17] Apache. Apache ServiceMix. Available at: http://servicemix.apache.org/home.html, last visited: May 2013.

[18] A. Arsanjani. Service-oriented modeling and architecture. *IBM developer works*, 2004.

[19] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. S3: A service-oriented reference architecture. *IT professional*, 9(3):10–17, 2007.

[20] Carliss Young Baldwin and Kim B Clark. *Design rules: The power of modularity*, volume 1. mIt Press, 2000.

[21] Montserrat Batet, David Isern, Lucas Marin, Sergio Martinez, Antonio Moreno, David Sanchez, Aida Valls, and Karina Gibert. Knowledge-driven delivery of home care services. *Journal of Intelligent Information Systems*, pages 1–36, 2010.

[22] Peter Baum, Fabienne Abadie, Francisco Villanueva Lupiañez, Ioannis Maghiros, Elena Villalba Mora, and Maria Bernarda Zamora Talaya. Market developments–remote patient monitoring, treatment, telecare, fitness/wellness and mhealth. 2013.

[23] George W Beeler. Hl7 version 3 - an object-oriented methodology for collaborative standards development. *International Journal of Medical Informatics*, 48(1):151–161, 1998.

[24] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *Advanced Information Systems Engineering*, pages 415–429. Springer, 2005.

[25] N. Bevan, J. Kirakowski, and J Maissel. What is Usability? In *Human Aspects in Computing: Design and Use of Interactive Systems with Terminals*, pages 651–655. Elsevier, 1991.

[26] A. Bottaro, E. Simon, S. Seyvoz, and A. Gerodolle. Dynamic web services on a home service platform. In *Proc. 22nd Int. Conf. Advanced Information Networking and Applications*, pages 378–385, 2008.

[27] John B. Bowles and C. Enrique Peláez. Fuzzy logic prioritization of failures in a system failure mode, effects and criticality analysis. *Reliability Engineering & System Safety*, 50(2):203–213, 1995.

[28] BPMN. Business process model and notation, version 2.0. Available at: http://www.omg.org/spec/BPMN/2.0/PDF/, last visited: May 2013.

[29] Gerry Brueckner and Diane Face. Implementing an effective home care risk management program. *Perspectives in Healthcare Risk Management*, 9(4):25–28, 1989.

[30] Christoph Bussler, Dieter Fensel, and Alexander Maedche. A conceptual architecture for semantic web enabled web services. *ACM Sigmod Record*, 31(4):24–29, 2002.

[31] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming Shan. Adaptive and Dynamic Service Composition in eFlow. In *Advanced Information Systems Engineering*, volume 1789, pages 13–31. Springer Berlin / Heidelberg, 2000.

[32] Marie Chan, Daniel Esteve, Christophe Escriba, and Eric Campo. A review of smart homes present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, July 2008.

[33] Anis Charfi and Mira Mezini. Aspect-oriented web service composition with ao4bpel. In *Web Services*, pages 168–182. Springer, 2004.

[34] Anis Charfi and Mira Mezini. Hybrid Web Service Composition: Business Processes Meet Business Rules. In *2nd Int. Conf. on Service oriented computing*, ICSOC '04, pages 30–38. ACM, 2004.

[35] Anis Charfi and Mira Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web*, 10:309–344, 2007.

[36] Betty Cheng and et al. *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009.

[37] CLEAR. The Clinical Leading Environment for the Assessment of Rehabilitation protocols in home care, 2007. Available at: http://www.habiliseurope.eu/?q=node/5, last visited: January 2013.

[38] Cordys. Cordys BPM. Available at: http://www.cordys.com/, last visited: May 2013.

[39] Patricia Dockhorn Costa, Giancarlo Guizzardi, Joao Paulo A Almeida, Luis Ferreira Pires, and Marten van Sinderen. Situations in conceptual modeling of context. In *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, page 6. IEEE Computer Society, 2006.

[40] Francisco Curbera, Matthew J Duftler, Rania Khalaf, WA Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Colombo: Lightweight middleware for service-oriented computing. *IBM Systems Journal*, 44(4):799–820, 2005.

[41] D. Garlan and R. Allen and J. Ockerbloom. Architectural mismatch: Why reuse is *still* so hard. *IEEE Software*, 26(4):66–69, July-August 2009.

[42] Paul De Clercq, Katharina Kaiser, and Arie Hasman. Computer-Interpretable Guideline formalisms. *Studies in health technology and informatics*, 139:22–43, 2008.

[43] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[44] Anind K Dey and Gregory D Abowd. The context toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for wearable and pervasive computing*, pages 431–441, 2000.

[45] P. Dockhorn Costa, L. Ferreira Pires, and Marten van Sinderen. Architectural patterns for context-aware services platforms. In *2nd International Workshop on Ubiquitous Computing*, pages 3–18. INSTICC Press, 2005.

[46] Markus Döhring, Birgit Zimmermann, and Eicke Godehardt. Extended Workflow Flexibility using Rule-Based Adaptation Patterns with Eventing Semantics. In *Informatik2010 Service Science*, pages 216—-226. GI e.V., 2010.

[47] Dreaming. elDeRly-friEndly Alarm handling and MonitorING. Available at: http://www.dreaming-project.org/, last visited: January 2013.

[48] Duc Viet Bui and Maria-Eugenia Iacob and Marten van Sinderen and Alireza Zarghami. Achieving flexible process interoperability in the homecare domain through aspect-oriented service composition. In *The Fifth International IFIP Working Conference on Enterprise Interoperability (IWEI 2013)*, pages 50–64. Springer Berlin Heidelberg, March 2013.

[49] J.H. Duffus, S.S. Brown, and N.A.G.G. Fernicola. Glossary for chemists of terms used in toxicology. *International Union of Pure and Applied Chemistry*, 65:2003–2122, 1993.

[50] Eclipse. Eclipse Web Tools. Available at: http://www.eclipse.org/webtools/, last visited: May 2013.

[51] epSOS. Smart Open Services for European Patients, 2008. Available at: http://www.epsos.eu/, last visited: January 2013.

[52] Ernest Friedman-Hill. Jess. Available at: http://www.jboss.org/drools/drools-expert.html, last visited: May 2013.

[53] European Commission. Ageing well in the information society - an i2010 initiative - action plan on information and communication technologies and ageing. Technical report, EU, 2007.

[54] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*, volume 18. Springer Heidelberg, 2007.

[55] Juan A. Fraile, Javier Bajo, Ajith Abraham, and Juan M. Corchado. HoCaMA: Home Care Hybrid Multiagent Architecture. In *Pervasive Computing*, Computer Communications and Networks, pages 259–285. Springer London, 2010.

[56] Douglas B. Fridsma and Jan Thomsen. Representing Medical Protocols for Organizational Simulation: An Information-Processing Approach. *Computational and Mathematical Organization Theory*, 4:71–95, 1998.

[57] Erik Frøkjær, Morten Hertzum, and Kasper Hornbæk. Measuring usability: are effectiveness, efficiency, and satisfaction really correlated? In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 345–352. ACM, 2000.

[58] Katrin Gabner and Michael Conrad. ICT enabled independent living for elderly, A status-quo analysis on products and the research landscape in the field of Ambient Assisted Living in EU-27. prepared by VDI/VDE Innovation und Technik GmbH, 2010.

[59] T. Gao, D. Greenspan, M. Welsh, R. Juang, and A. Alm. Vital signs monitoring and patient tracking over a wireless network. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 102–105. IEEE, 2006.

[60] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Software*, 12(6):17–26, November-December 1995.

[61] Pallapa Gautham and Das Sajal. Challenges of designing privacy enhanced context-aware middleware for assisted healthcare. In *Smart Homes and Health Telematics*, volume 5120 of *LNCS*, pages 200–207. Springer Berlin / Heidelberg, 2008.

[62] GlassFish. http://glassfish.java.net/. Available at: http://glassfish.java.net/, last visited: May 2013.

[63] V. Gomoi and V. Stoicu-Tivadar. A new method in automatic generation of medical protocols using artificial intelligence tools and a data manager. In *Proc. Int Computational Cybernetics and Technical Informatics (ICCC-CONTI) Joint Conf*, pages 243–246, 2010.

[64] Colin Gordon and Jens Pihlkjaer Christensen. Health Telematics for Clinical Guidelines and Protocols. IOS Press, 1995.

[65] P. D. Gray, T. McBryan, N. Hine, C. J. Martin, N. Gil, M. Wolters, N. Mayo, K. J. Turner, L. S. Docherty, F. Wang, and M. Kolberg. A scalable home care system infrastructure supporting domiciliary care. Technical report, Department of Computing Science and Mathematics University of Stirling, 2007.

[66] W3C Working Group et al. Web services glossary. *W3C Working Group Note, Website: http://www. w3. org/TR/ws-gloss*, 2004.

[67] T. Gu, H.K. Pung, and D.Q. Zhang. Toward an osgi-based infrastructure for context-aware applications. *Pervasive Computing, IEEE*, 3(4):66 – 74, 2004.

[68] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference*, volume 2004, pages 270–275, 2004.

[69] Charles B. Haley, Jonathan D. Moffett, Robin Laney, and Bashar Nuseibeh. Arguing security: validating security requirements using structured argumentation. In *Third Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05)*, 2005.

[70] Reinhold Haux, Elske Ammenwerth, Werner Herzog, Petra Knaup, et al. Health care in the information society. a prognosis for the year 2013. *International Journal of Medical Informatics*, 66(1):3–22, 2002.

[71] E. Hollnagel. *Human reliability analysis: Context and control*. Academic Press London, 1993.

[72] Andreas Holzinger. User-centered interface design for disabled and elderly people: First experiences with designing a patient communication system (pacosy). In *Computers helping people with special needs*, pages 33–40. Springer, 2002.

[73] IBM. IBM DB2 database software. Available at: http://www-01.ibm.com/software/data/db2/, last visited: May 2013.

[74] IBM. WebSphere Lombardi Edition. Available at: http://www-01.ibm.com/software/integration/lombardi-edition/, last visited: May 2013.

[75] IBM. WebSphere Lombardi Edition. Available at: http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/index.jsp, last visited: May 2013.

[76] IEEE. Institute of electrical and electronics engineers: Ieee standard computer dictionary: A compilation of ieee standard computer glossaries, New York, NY, 1990.

[77] Informatica. ActiveVOS. Available at: http://www.activevos.com/, last visited: May 2013.

[78] innospense. Medido Farmaceutische Telezorg. Available at: http://www.innospense.com, last visited: May 2013.

[79] T. Ishimatsu, N. Leveson, J. Thomas, M. Katahira, Y. Miyamoto, and H. Nakao. Modeling and hazard analysis using stpa. In *Conference of the International Association for the Advancement of Space Safety, Huntsville, Alabama*, 2010.

[80] ISO. Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability. International Standard 9241-11, 1998.

[81] ISO/IEC. Information technology, Security techniques, Guidelines for the management of IT Security: Concepts and models. Intl. Std. 13335-1, 2004.

[82] M.A. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[83] François Jammes and Harm Smit. Service-oriented paradigms in industrial automation. *Industrial Informatics, IEEE Transactions on*, 1(1):62–70, 2005.

[84] JBoss Community. Drools Expert. Available at: http://www.jboss.org/drools/drools-expert.html, last visited: May 2013.

[85] JBoss Community. Drools Flow. Available at: http://www.jboss.org/drools/drools-flow, last visited: May 2013.

[86] S. Johnston. Uml 2.0 profile for software services. *IBM developerWorks http://www.ibm. com/developerworks/rational/library/05/419_soa*, 2005.

[87] Swaroop Kalasapur, Mohan Kumar, and Behrooz A Shirazi. Dynamic service composition in pervasive computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, 2007.

[88] R. Kennedy and B. Kirwan. Development of a Hazard and Operability-based method for identifying safety management vulnerabilities in high risk systems. *Safety Science*, 30(3):249–274, December 1998.

[89] Nigel King and Christine Horrocks. *Interviews in qualitative research*. SAGE Publications Limited, 2010.

[90] Thomas Kleinberger, Martin Becker, Eric Ras, Andreas Holzinger, and Paul Müller. Ambient intelligence in assisted living: enable elderly people to handle future interfaces. In *Universal access in human-computer interaction. Ambient interaction*, pages 103–112. Springer, 2007.

[91] I. Korhonen, J. Parkka, and M. Van Gils. Health monitoring in the home of the future. *Engineering in Medicine and Biology Magazine, IEEE*, 22(3):66 – 73, may-june 2003.

[92] K. Kosanke. Iso standards for interoperability: a comparison. *Interoperability of Enterprise Software and Applications*, pages 55–64, 2006.

[93] Richard A Krueger and Mary Anne Casey. *Focus groups: A practical guide for applied research*. SAGE Publications, Incorporated, 2008.

[94] Ariella Lang, Nancy Edwards, and Andrea Fleiszer. Safety in home care: a broadened perspective of patient safety. *Intl. J. for Quality in Health Care*, 20:130–135, 2008.

[95] G. Loniewski, E.L. Ramon, S. Walderhaug, S. Martinez Franco, J.J. Cubillos Esteve, and E.S. Marco. Data Management in an Intelligent Environment for Cognitive Disabled and Elderly People. *Electronic Healthcare*, pages 50–57, 2009.

[96] Shiyong Lu, Ming Dong, and Farshad Fotouhi. The semantic web: opportunities and challenges for next-generation web applications. *Information research*, 7(4):7–4, 2002.

[97] Norbert Malanowski, Rukiye Ozcivelek, and Marcelino Cabrera. Active Ageing and Independent Living Services, The Role of Information and Communication Technology. European Communitiy, 2008. Available at: http://www.umic.pt/images/stories/publicacoes2/JRC41496.pdf.

[98] MATCH. Mobilising Advanced Technologies for Care at Home, 2005-2012. Available at: http://www.match-project.org.uk/main/main.html, last visited: May 2013.

[99] Tony Mcbryan and Phil Gray. Interactive systems. design, specification, and verification. pages 167–180. Springer-Verlag, Berlin, Heidelberg, 2008.

[100] Tony McBryan, Marilyn R. McGee-Lennon, and Phil Gray. An integrated approach to supporting interaction evolution in home care systems. In *PETRA '08: Proceedings of the 1st Int. Conf. on Pervasive Technologies Related to Assistive Environments*, pages 1–8, New York, NY, USA, 2008. ACM.

[101] Marilyn Rose McGee-Lennon. Requirements engineering for home care technology. In *26th Annual SIGCHI Conf. on Human Factors in Computing Systems*, pages 1439–1442, 2008.

[102] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. *Knowledge and Data Engineering, IEEE Transactions on*, 17(7):954 – 968, 2005.

[103] S. Meystre. The current state of telemonitoring: a comment on the literature. *Telemedicine Journal & e-Health*, 11(1):63–69, 2005.

[104] M. Mikalsen, S. Hanke, T. Fuxreiter, S. Walderhaug, L. Wienhofen, and N. Trondheim. Interoperability services in the MPOWER Ambient Assisted Living platform. *Stud Health Technol Inform*, 150:366–70, 2009.

[105] Ministers, local government, NHS, social care, professional and regulatory organisations. Putting people first: a shared vision and commitment to the transformation of adult social care. Technical report, NHS Direct, 2007.

[106] MobiHealth. MobiHealth. Available at: http://www.eclipse.org/webtools/, last visited: May 2013.

[107] Mohammad Zarifi Eslami and Alireza Zarghami and Brahmananda Sapkota and Marten van Sinderen.

Flexible Homecare Application Personalization and Integration Using Pattern-Based Service Tailoring: Supporting Independent Living of Elderly with IT. In *11th IEEE International Conference on Computer and Information Technology (CIT)*, pages 467–474. IEEE Computer Society, 2011.

[108] Mohammad Zarifi Eslami and Brahmananda Sapkota and Alireza Zarghami and Eelco Vriezekolk and Marten van Sinderen and Roel Wieringa. Risk Identification of Tailorable Context-aware Systems: a Case Study and Lessons Learned. In *Proceedings of the CAiSE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 855 of *CEUR Workshop Proceedings*, pages 40–49. CEUR-WS.org, 2012.

[109] G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, 2004.

[110] MPOWER. Middleware Platform for eMPOWERing cognitive disabled and elderly, 2006-2009. Available at: http://www.sintef.no/Projectweb/MPOWER.

[111] Meenakshi Nagarajan, Kunal Verma, Amit P Sheth, John Miller, and Jon Lathem. Semantic interoperability of web services-challenges and experiences. In *Web Services, 2006. ICWS'06. International Conference on*, pages 373–382. IEEE, 2006.

[112] O. Nee, A. Hein, T. Gorath, N. Hulsmann, G. B. Laleci, M. Yuksel, M. Olduz, I. Tasyurt, U. Orhan, A. Dogac, A. Fruntelata, S. Ghiorghe, and R. Ludwig. Saphire: intelligent healthcare monitoring based on semantic interoperability platform: pilot applications. *IET Communications*, 2(2):192–201, 2008.

[113] V. Nunes Leal Franqueira, T. T. Tun, Y. Yu, R. J. Wieringa, and B. Nuseibeh. Risk and argument: A risk-based argumentation method for practical security. In *Proceedings of the 19th IEEE International Requirements Engineering Conference, Trento, Italy*, pages 239–248, USA, June 2011. IEEE Computer Society.

[114] OASIS. Web Services Business Process Execution Language Version 2.0. Available at: https://www.oasis-open.org/committees/download.php/23964/, last visited: May 2013.

[115] Orbis. Orbis medisch en zorgconcern. Available at: http://www.orbisconcern.nl/, last visited: May 2013.

[116] Carsten Orwat, Andreas Graefe, and Timm Faulwasser. Towards pervasive computing in health care–a literature review. *BMC Medical Informatics and Decision Making*, 8(1):26, 2008.

[117] Mike P Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3):389–415, 2007.

[118] S.J. Pereira, G. Lee, and J. Howard. A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. Technical report, DTIC Document, 2006.

[119] Shankar R Ponnekanti and Armando Fox. Interoperability among independently evolving web services. In *Middleware 2004*, pages 331–351. Springer, 2004.

[120] Hung Keng Pung, Tao Gu, Wenwei Xue, Paulito P Palmes, Jian Zhu, Wen Long Ng, Chee Weng Tang, and Nguyen Hoang Chung. Context-aware middleware for pervasive elderly homecare. *Selected Areas in Communications, IEEE Journal on*, 27(4):510–524, 2009.

[121] Silvana Quaglini, Mario Stefanelli, Giordano Lanzola, Vincenzo Caporusso, and Silvia Panzarasa. Flexible guideline-based patient careflow systems. *Artificial Intelligence in Medicine*, 22(1):65–80, 2001.

[122] Fano Ramparany and Laurent Vercouter. Flexible composition of smart device services. In *In: The 2005 International Conference on Pervasive Systems and Computing(PSC-05), Las Vegas*, pages 27–30, 2005.

[123] Jinghai Rao and Xiaomeng Su. A Survey of Automated Web Service Composition Methods. In *Semantic Web Services and Web Process Composition*, volume 3387, pages 43–54. Springer Berlin / Heidelberg, 2005.

[124] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Jorge Cardoso and Amit Sheth, editors, *Semantic Web Services and Web Process Composition*, volume 3387 of *LNCS*, pages 43–54. Springer Berlin / Heidelberg, 2005.

[125] F. Redmill, M.F. Chudleigh, and J.R. Catmur. Principles underlying a guideline for applying HAZOP to programmable electronic systems. *Reliability Engineering & System Safety*, 55(3):283 – 293, 1997.

[126] Vincent Rialle, Florence Duchene, Norbert Noury, Lionel Bajolle, and Jacques Demongeot. Health smart home: information technology for patients at home. *Telemedicine Journal and E-Health*, 8(4):395–409, 2002.

[127] Daniel Ronzani. The battle of concepts: Ubiquitous computing, pervasive computing and ambient intelligence in mass media. *Ubiquitous Computing and Communication Journal*, 4(2):9–19, 2009.

[128] F. Rosenberg and S. Dustdar. Business Rules Integration in BPEL - a Service-oriented Approach. In *7th IEEE International Conf. on E-Commerce Technology*, pages 476–479, 2005.

[129] B. Sapkota, C.H. Asuncion, M. Iacob, and M. van Sinderen. A Simple Solution for Information Sharing in Hybrid Web Service Composition. In *15th IEEE Int. Conf. on Enterprise Distributed Object Computing Conference*, pages 235 –244, 29 2011-sept. 2 2011.

[130] Richard E Schantz and Douglas C Schmidt. Middleware for distributed systems: Evolving the common structure for network-centric applications. *Encyclopedia of Software Engineering*, 1, 2002.

[131] R. Seater, D. Jackson, and R. Gheyi. Requirement progression in problem frames: deriving specifications from requirements. *Requirements Engineering*, 12(2):77–102, 2007.

[132] B. Shackel. The concept of usability. In *Visual display terminals: Usability issues and health concerns*, pages 45–87. Englewood Cliffs, NJ:Prentice-Hall, 1984.

[133] S. P. Sivasubramanian, E. Ilavarasan, and G. Vadivelou. Dynamic web service composition: Challenges and techniques. In *Proc. Int. Conf. Intelligent Agent & Multi-Agent Systems IAMA 2009*, pages 1–8, 2009.

[134] Bianying Song, K.-H. Wolf, M. Gietzelt, O. Al Scharaa, U. Tegtbur, R. Haux, and M. Marschollek. Decision support for teletraining of COPD patients. In *Proc. 3rd Int. Conf. Pervasive Computing Technologies for Healthcare PervasiveHealth 2009*, pages 1–6, 2009.

[135] Bea Steenbekkers and Marlies van der Linden. Ageing in Europe: Trends and Challenges. In *ELDERATHOME: The prerequisites of the elderly for living at home: Criteria for dwellings, surroundings and facilities: Final report*. TTS Institute's Publication 393, 2004.

[136] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[137] H. Sun, V. De Florio, N. Gui, and C. Blondia. Promises and challenges of ambient assisted living systems. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1201–1207. IEEE, 2009.

[138] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27–46, 2003.

[139] T-SENIORITY. Expanding the benefits of Information Society to Older People through digital TV channels, 2006. Available at: http://tseniority.idieikon.com/index.php/lang-en/the-project, last visited: January 2013.

[140] The University of Twente. Biomedical Signals And Systems. Available at: http://www.utwente.nl/ewi/bss/, last visited: May 2013.

[141] The University of Twente. Information Systems Group. Available at: http://www.utwente.nl/ewi/is/, last visited: May 2013.

[142] U-care team. U-care project. Available at: http://www.utwente.nl/ewi/ucare/, last visited: May 2013.

[143] Ubuntu. Ubuntu 10.04.4 LTS (Lucid Lynx). Available at: http://releases.ubuntu.com/lucid/, last visited: May 2013.

[144] UN-ISDR. Terminology on Disaster Risk Reduction. Geneva, 2009.

[145] Wil MP Van Der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management*, pages 161–183. Springer, 2000.

[146] Axel Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262. IEEE, 2001.

[147] David Vose. *Risk analysis: a quantitative guide*. Wiley, 2008.

[148] VTT Technical Research Centre of Finland. VantagePoint. Available at: http://www.vtt.fi/proj/vantagepoint/?lang=en, last visited: May 2013.

[149] Maja Vukovic, Evangelos Kotsovinos, and Peter Robinson. An architecture for rapid, on-demand service composition. *Service Oriented Computing and Applications*, 1:197–212, 2007.

[150] W3C. Simple Object Access Protocol (SOAP) 1.1 . Available at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, last visited: May 2013.

[151] W3C. Web Services Description Language (WSDL) 1.1. Available at: http://www.w3.org/TR/wsdl, last visited: May 2013.

[152] Nanbor Wang, Douglas C Schmidt, Aniruddha Gokhale, Christopher D Gill, Balachandran Natarajan, Craig Rodrigues, Joseph P Loyall, and Richard E Schantz. Total quality of service provisioning in middleware and applications. *Microprocessors and Microsystems*, 27(2):45–54, 2003.

[153] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.

[154] C.C. White, D. Fang, E.-H. Kim, W.B. Lober, and Y. Kim. Improving healthcare quality through distributed diagnosis and home healthcare. In *1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare (D2H2).*, pages 168 –172, april 2006.

[155] R.J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate, and the UML.* Morgan Kaufmann, 2003.

[156] Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, pages 8:1–8:12. ACM, 2009.

[157] Roel Wieringa. A unified checklist for observational and experimental research in software engineering (version 1), March 2012.

[158] Roel Wieringa and Ayse Morali. Technical Action Research as a Validation Method in Information Systems Design Science. In *7th International Conference, DESRIST*, volume 7286 of *Lecture Notes in Computer Science*, pages 220–238. Springer, 2012.

[159] Wikipedia. Bluetooth. Available at: http://en.wikipedia.org/wiki/Bluetooth, last visited: May 2013.

[160] Wikipedia. General Packet Radio Service. Available at: http://en.wikipedia.org/wiki/General_Packet_Radio_Service, last visited: May 2013.

[161] Wikipedia. Hypertext Transfer Protocol (HTTP). Available at: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, last visited: May 2013.

[162] Wikipedia. Short Message Service(SMS). Available at: http://en.wikipedia.org/wiki/Short_Message_Service, last visited: May 2013.

[163] Wikipedia. Windows Server 2008. Available at: http://en.wikipedia.org/wiki/Windows_Server_2008, last visited: May 2013.

[164] Yuping Yang, Fiona Mahon, M. Williams, and Tom Pfeifer. Context-aware dynamic personalised service re-composition in a pervasive service environment. In *Ubiquitous Intelligence and Computing*, volume 4159, pages 724–735. Springer Berlin / Heidelberg, 2006.

[165] Alireza Zarghami, Brahmananda Sapkota, Mohammad Zarifi Eslami, and Marten van Sinderen. Decision as a Service: Separating Decision-making from Application Process Logic. In *The 16h IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 103–112. IEEE, 2012.

[166] Alireza Zarghami, Eelco Vriezekolk, Mohammad Zarifi Eslami, Marten van Sinderen, and Roel Wieringa. Assumption-based Risk Identification Method (ARM) in Dynamic Service Provisioning. In *The 21st IEEE International Requirements Engineering Conference (RE)*, pages 175–184. IEEE, July 2013.

[167] Alireza Zarghami, Mohammad Zarifi Eslami, and Marten van Sinderen. What Do Homecare Provider Stories Tell Us about Dynamicity? In *17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 44, pages 154–155. ICB Institut fur Informatik und Wirtschaftsinformatik, 2011.

[168] Mohammad Zarifi Eslami. *Service tailoring: a method and tool for user-centric creation of integrated IT-based homecare services to support independent living of elderly*. PhD thesis, Univ. of

Twente, Enschede, June 2013. SIKS Dissertation Series No. 2013-13.

[169] Mohammad Zarifi Eslami and Marten van Sinderen. Flexible home care automation adapting to the personal and evolving needs and situations of the patient. In *3rd International Conference on Pervasive Computing Technologies for Healthcare*, pages 1 –2, april 2009.

[170] Shuai Zhang, Sally I. McClean, Bryan W. Scotney, Xin Hong, Chris D. Nugent, and Maurice D. Mulvenna. Decision support for alzheimer's patients in smart homes. In *CBMS*, pages 236–241, 2008.

[171] Michael zur Muehlen, Marta Indulska, and Kai Kittel. Towards Integrated Modeling of Business Processes and Business Rules. In *19th Australasian Conference on Information Systems*, pages 690–699, 2008.

Author Publications

Alireza Zarghami, Eelco Vriezekolk, Mohammad Zarifi Eslami, Marten van Sinderen, and Roel Wieringa. Assumption-based Risk Identification Method (ARM) in Dynamic Service Provisioning. In *The 21st IEEE International Requirements Engineering Conference (RE)*, pages 175–184. IEEE, July 2013.

Alireza Zarghami, Mohammad Zarifi Eslami, Marten van Sinderen, and Roel Wieringa. Dynamic homecare service provisioning: A field test and its results. In *Process Support and Knowledge Representation in Health Care*. Springer, June 2013.

Duc Viet Bui, Maria-Eugenia Iacob, Marten van Sinderen, and Alireza Zarghami. Achieving flexible process interoperability in the homecare domain through aspect-oriented service composition. In *The Fifth International IFIP Working Conference on Enterprise Interoperability (IWEI 2013)*, pages 50–64. Springer Berlin Heidelberg, March 2013.

Mohammad Zarifi Eslami, Brahmananda Sapkota, Andrea Herrmann, Alireza Zarghami, Marten van Sinderen, and Roel Wieringa. Risk Driven Requirements Specification (RiDeRS) of IT-based Homecare Systems. In *Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, CEUR Workshop Proceedings. CEUR-WS.org, 2013.

Mohammad Zarifi Eslami, Alireza Zarghami, Marten van Sinderen, and Roel Wieringa. Care-giver Tailoring of IT-based Healthcare Services for Elderly at Home: A Field Test and its Results. In *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*. IEEE, 2013.

Alireza Zarghami and Brahmananda Sapkota and Mohammad Zarifi Eslami and Marten van Sinderen. Decision as a Service: Separating Decision-making from Application Process Logic. In *The 16h IEEE*

Int.l Enterprise Distributed Object Computing Conference (EDOC), pages 103–112. IEEE, 2012.

Mohammad Zarifi Eslami, Brahmananda Sapkota, Alireza Zarghami, Eelco Vriezekolk, Marten van Sinderen, and Roel Wieringa. Risk Identification of Tailorable Context-aware Systems: a Case Study and Lessons Learned. In *Proceedings of the CAiSE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 855 of *CEUR Workshop Proceedings*, pages 40–49. CEUR-WS.org, 2012.

Alireza Zarghami, Mohammad Zarifi Eslami, Brahmananda Sapkota, and Marten van Sinderen. Service Realization and Compositions Issues in the Homecare Domain. In *6th International Conference on Software and Data Technologies (ICSOFT), Volume 1*, pages 347–356. SciTePress, July 2011.

Alireza Zarghami, Mohammad Zarifi Eslami, Brahmananda Sapkota, and Marten van Sinderen. Toward Dynamic Service Provisioning in the Homecare Domain. In *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth), Workshop on Designing and Integrating Independent Living Technology*, pages 292–299. IEEE, May 2011.

Mohammad Zarifi Eslami, Alireza Zarghami, Brahmananda Sapkota, and Marten van Sinderen. Personalized Service Creation by Non-technical Users in the Homecare Domain. *Procedia CS*, 5:409–417, 2011.

Alireza Zarghami, Mohammad Zarifi Eslami, Brahmananda Sapkota, and Marten van Sinderen. Dynamic Homecare Service Provisioning Architecture. In *9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, 2011.

Alireza Zarghami, Mohammad Zarifi Eslami, and Marten van Sinderen. What Do Homecare Provider Stories Tell Us about Dynamicity? In *17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 44, pages 154–155. ICB Institut fur Informatik und Wirtschaftsinformatik, 2011.

Mohammad Zarifi Eslami, Alireza Zarghami, Brahmananda Sapkota, and Marten van Sinderen. Flexible Homecare Application Personalization and Integration Using Pattern-Based Service Tailoring: Supporting Independent Living of Elderly with IT. In *11th IEEE International Conference on Computer and Information Technology (CIT)*, pages 467–474. IEEE Computer Society, 2011.

Mohammad Zarifi Eslami, Alireza Zarghami, Brahmananda Sapkota, and Marten van Sinderen. Service Tailoring: Towards Personalized Homecare Services. In *4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC)*, pages 109–121. SciTePress, July 2010.

Soude Fazeli, Alireza Zarghami, Nima Dokoohaki, and Mihhail Matskin. Mechanizing social trust-aware recommenders with t-index augmented trustworthiness. In *Trust, Privacy and Security in Digital Business*, pages 202–213. Springer Berlin Heidelberg, 2010.

Soude Fazili, Alireza Zarghami, Nima Dokoohaki, and Mihhail Matskin. Elevating prediction accuracy n trust-aware collaborative filtering recommenders through t-index metric and toptrustee lists. *Journal of Emerging Technologies in Web Intelligence*, 2(4):300–309, 2010.

Alireza Zarghami, Soude Fazeli, Nima Dokoohaki, and Mihhail Matskin. Social trust-aware recommendation system: A t-index approach. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 03*, pages 85–90. IEEE Computer Society, 2009.

Appendix A

Questionnaires

Usability Survey of the UCare system (final questionnaires from the care-givers)

Please provide your opinions on the system by answering the questions below. You can also comment on each question if you think the question is not applicable or complete enough.

1. Which things you can **do** with the system that you could not do before?
   - Answer:
   - Comment on the answer (optional):
2. Which things you can**not do** with the system that you could do before?
   - Answer:
   - Comment on the answer (optional):
3. Using the system you can schedule, personalize and measure four care tasks namely: blood pressure, weight, oxygen saturation and medication intake. Can you see any other tasks which could be performed using the system which is not supported now?
   - Answer:
   - Comment on the answer (optional):
4. Have you ever get upset (frustrated, angry) using the system? If yes, please describe the situation.
   - Answer:
   - Comment on the answer (optional):
5. Have you ever been pleasantly surprised when you were using the system? A situation that you didn't expect. If yes, please describe the situation.
   - Answer:
   - Comment on the answer (optional):
6. Does using the system save your time? In which task(s) and in which way?
   - Answer:
   - Comment on the answer (optional):
7. Does using the system cause some tasks to take more time than before? In which task and in which way?
   - Answer:
   - Comment on the answer (optional):
8. Does using the system make things easier? In which task(s) and in which way?

Figure A-1 Usability questionnaire for the second series of experiments, page 1

- Answer:
- Comment on the answer (optional):

9. Does using the system make things difficult? In which task(s) and in which way?
   - Answer:
   - Comment on the answer (optional):
10. Does using the system increase the **quality of care**? In which way?
   - Answer:
   - Comment on the answer (optional):
11. Does using the system decrease the **quality of care**? In which way?
   - Answer:
   - Comment on the answer (optional):
12. Does using the system increase the **quality of life** of elderly? In which way?
   - Answer:
   - Comment on the answer (optional):
13. Does using the system decrease the **quality of life** of elderly? In which way?
   - Answer:
   - Comment on the answer (optional):
14. In which situations do you think the IT-based homecare system could be used successfully in practice? Please give examples.
   - Answer:
   - Comment on the answer (optional):
15. In which situations do you think a system like this **should not** be used? Please give examples
   - Answer:
   - Comment on the answer (optional):
16. Does the system remind the care-receivers on time to measure their vital signs or to take their medications?
   - Answer:
   - Comment on the answer (optional):

Figure A-2 Usability questionnaire for the second series of experiments, page 2

17. After measuring the vital signs, does the system show the measured values on the Tablet PC in a reasonable time?
    - Answer:
    - Comment on the answer (optional):
18. Does the system stop sending reminders after the care-receiver has measured his/her vital signs?
    - Answer:
    - Comment on the answer (optional):
19. Does the system send an alert on time when the measured vital signs are too high/low?
    - Answer:
    - Comment on the answer (optional):
20. You use and configure different devices and applications (e.g., medicine dispenser, blood pressure measurement device) from different service providers. Did you notice that they are provided by different companies?
    - Answer:
    - Comment on the answer (optional):
21. In case your answer to question 5 is “yes”, how did you notice?
    - Answer:
    - Comment on the answer (optional):
22. The system uses both manual medication intake and automatic medication dispenser. Does the switching between different medication approaches change the rest of medication monitoring application? (e.g., reminder)
    - Answer:
    - Comment on the answer (optional):
23. Did you find it useful that you can choose between manual medication intake and automatic medication dispenser?
    - Answer:
    - Comment on the answer (optional):
24. When you tailor the application (e.g., changing number of reminder), does the system immediately update its behavior?
    - Answer:
    - Comment on the answer (optional):

Figure A-3 Usability questionnaire for the second series of experiments, page 3

SIKS Dissertation Series

====
1998
====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games
1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

====
2000
====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)

Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005
====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based
Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in
Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics
meets pragmatics

2005-15 Tibor Bosse (VU)

Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment,
and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of
Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured
Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated
media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems:
a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy:
a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)

Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramï¿½rez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====
2008
====

2008-01 Katalin Boer-Sorbï¿½n (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms
for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of
Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van
elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using
Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users,
and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and
Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
perating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management:
An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and

Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children 2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of

Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

====
2011
====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues 2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans
On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-
Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning
and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)

Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture
for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues:
design aspects influencing interaction quality

====
2012
====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance
under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)

Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Grkk (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)
Risk-aware Design of Value and Coordination Networks

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn
-

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)
Ontology driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

====
2013
====

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD)
Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns

2013-06 Romulo Gonzalves(CWI)
The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UT)
Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU)
Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT)
A Unified Modeling Framework for Service Design

2013-11 Evangelos Pournaras (TUD)
Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Maryam Razavian (VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Zarifi Eslami (UT)
A Method and Tool for User-centric Creation of Integrated IT-based Homecare Services to Support Independent Living of Elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuisen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UVA)
Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva (UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnes Berendsen (UM)
Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning

Samenvatting

Het gebruik van IT in hulpverlening aan thuiswonende ouderen (zorgontvangers) is aangedragen als een veelbelovende aanpak voor het probleem van vergrijzing. Met de opkomst van dienstverleners voor thuiszorgtoepassingen kan een thuiszorgsysteem worden gezien als een verzameling gekoppelde diensten. De kosten van applicatieontwikkeling kunnen worden verlaagd door thuiszorgtoepassingen te configureren en samen te stellen uit bestaande thuiszorgdiensten. Dit concept wordt extra aantrekkelijk als dienstverlening dynamisch is, wat betekent dat toepassingen hun gedrag kunnen aanpassen aan veranderingen in hun omgeving met geen of weinig mankracht.

Dit proefschrift beschrijft een Dynamisch Thuiszorg Dienstverleningsplatorm (DTD platform) dat contextuele veranderingen in thuiszorg op een effectieve en efficiÃ«nte wijze kan adresseren. Met dynamische dienstverlening kan de configuratie van een samengestelde applicatie worden aangepast. Dat kan automatisch gebeuren terwijl de applicatie actief is (adaptieve dienstsamenstelling), door eindgebruikers zoals verpleegkundigen (wij noemen dit aanpasbare compositie van diensten) of door een programmeur (wij noemen dit ontwikkelbare compositie van diensten). Het voorgestelde DTD platform maakt adaptieve, aanpasbare en ontwikkelbare dienstverlening mogelijk op het gebied van thuiszorg.

Om dit te faciliteren wordt een hybride aanpak van compositie van diensten voorgesteld, waarin de kern van de toepassing, die tamelijk stabiel is, wordt beschreven in termen van processen, terwijl regels worden gebruikt om de voorwaarden en beperkingen vast te leggen om het gedrag van de toepassing aan te passen. De regels worden dan ontsloten als een $beslissingsdienst$ die door het proces kan worden gebruikt om adaptieve besluiten te nemen over de omstandigheden tijdens uitvoering.

Om de bruikbaarheid van dit concept aan te tonen hebben we een software prototype van ons platform ontwikkeld. Dit prototype is vervolgens gebruikt in een praktijktest waarin we twee experimenten hebben uitgevoerd in een Nederlandse instelling voor thuiszorg om de aanpak te valideren. Deze validatie omvat zowel objectieve als

subjectieve metingen. Het kunnen combineren van objectieve en subjectieve metingen is nuttig om te weten te komen welk niveau van effectiviteit en efficiëntie acceptabel is in thuiszorg. We hebben ook getracht verklaringen te vinden voor de uitkomst van de metingen, waardoor we konden begrijpen welke delen van onze aanpak verbeterd moeten worden.

Tijdens de praktijktest is het DTD platform gedurende vier maanden dagelijks gebruikt met in totaal meer dan 400.000 transacties in de infrastructuur en toepassingsdiensten. Het doel van de test was om de bruikbaarheid van het DTD platform te bestuderen om zo contextuele veranderingen in thuiszorg te adresseren in termen van (a) *effectiviteit*, (b) *efficiëntie*, en (c) *tevredenheid*, zowel subjectief als objectief.

Tijdens ons eerste experiment ontdekten we dat, hoewel de toepassingsdiensten die feitelijk geleverd werden door de dienstverleners voldeden aan de eisen van de eindgebruikers, de architecturen van de dienstverleners niet op elkaar aansloten vanwege onuitgesproken aannames. Daarom hebben we een Aanname-gebaseerde Risicoinventarisatiemethode (ARM) geïntroduceerd. De ARM methode heeft ons geholpen om diverse risico's te identificeren voordat het DTD platform gebruikt werd in ons tweede experiment.

In de praktijktest observeerden we dat de adaptiviteit van de thuiszorgapplicaties overeenkwam met de verwachtingen van de eindgebruikers (zorgontvangers en verpleegkundigen), tenminste tijdens het tweede experiment. De aanpasbaarheid van de thuiszorgapplicaties kwam ook overeen met de verwachtingen van het verpleegkundigen, met uitzondering van een specifiek type thuiszorgapplicatie. We zagen ook dat de ontwikkelbaarheid van thuiszorgapplicaties overeen kwam met de verwachtingen van de programmeurs. Dit werd voornamelijk mogelijk gemaakt door het gebruik van de beslissingsdienst. Onze praktijktest toonde aan dat het gebruik van de beslissingsdienst de ontwikkelbaarheid verbeter terwijl de kosten in termen van tijd en datacommunicatie tamelijk klein zijn.

Onze conclusie uit de praktijktest is dat het DTD platform bruikbaar is. We hebben echter het voorgestelde DTD platform slechts geëvalueerd in èèn zorginstelling in Nederland waarin zorgontvangers in hun zorgappartement wonen. Evaluatie van het platform in andere situaties (bijvoorbeeld in de situatie waarin zorgontvangers in hun eigen huis wonen en op afstand zorg ontvangen) kan andere resultaten geven. Bovendien moet het platform worden geëvalueerd met andere thuiszorgtoepassingen en hun vereiste toepassingsdiensten.

Glossary

adaptive application is an application that monitors foreseen observable changes and reacts to them based on predefined application logic. 25

application service is a concrete service that provides a set of functionalities which can be accessed by standard communication protocols such as SOAP. 22

care home is either a private home located outside of a care center or a unit located inside a care center. 29

care-giver Expert and volunteer who provides care and social services to elderly. 29

care-receiver Elderly who lives in a care home and receives care services from their care-givers. 29

composite application is an application which is composed of application services provided by possibly different, economically independent service providers. 22

data model is a data schema that defines the structure of the configuration parameters which are used by the decision rules. 88, 102

decision rule is a application-logic decision making rule that determines how to select, configure, or compse the application services with respect to runtime contextual situation. 80

domain expert is a person who has the domain knowledge and can define the behaviour of the application by assigning values to the configuration parameters of the service plan. 27

Dynamic Homecare Service Provisioning (DHSP) platform is a dynamic service provisioning platform to address the dynamicity requirements of the homecare domain including the required SBBs for homecare applications. 29

dynamic service provisioning is a type of service provisioning in which composite applications can update their behaviours with respect to the contextual changes without or with minimum

manpower. 22

dynamic service provisioning platform is an adaptive, tailorable and evolvable application service provisioning platform. The platform should support applications for adaptivity, end-users for the tailorability of the applications and programmers for the evolvability of the applications. 26

end-user is a person who use the composite applications running on top of the platform.. 26

evolvable application is an application that facilitates the manual update of application logic (i.e., reducing required manpower and system resources) to address unforeseen changes. 25

foreseen changes is a contextual change which is known by application programmer at design time, a.k.a. , a prior and she or he can define a response of application to that contextual changes by defining the application logic. 23

homecare system A system includes platforms, services, devices, data and networks that are required to support independent living of elderly. 29

infrastructure service is an application-scenario independent service which is used to either select, configure or compose the application services. 26

observable changes is a contextual change which can be monitored through physical sensors (i.e., non-symbolic interface). 23

orchestration pattern is part of a service plan and determines how the SBBs of a service plan and consequently, the selected application services are composed. A service plan can have several orchestration patterns. 80

programmer is a person with IT-knowledge who can do arbitrary IT-specific tasks to define or modify the application logic. 27

relevant contextual change is a contextual change, if it occurs, makes the current behaviour of a composite application undesired for the end-users of that application. 22

SBB defines a set of functionalities in the abstract level that can be implemented by alternative application services. 26, 28

service plan consists of one or more service building blocks and describes the configuration and composition of instances of these service building blocks with respect to run-time circumstances. 26

tailorable application is an application that can not observe and thus not monitor changes (this is done by the end-user), but can adapt its behaviour based on reconfiguration by the end-user. 25

tailoring platform is a platform that takes care of the service plan creation and tailoring, and eventually deploys the service plan to the provisioning platform for the execution. 26