MASTER THESIS

A SOA-BASED ARCHITECTURE FOR DEALING WITH D&D REQUIREMENTS IN THE HOMECARE DOMAIN

DUC BUI VIET

BUSINESS INFORMATION TECHNOLOGY

EXAMINATION COMMITTEE
Dr. Marten van Sinderen
Dr. Maria- Eugenia Iacob
Alireza Zarghami

AUGUST 8, 2011

UNIVERSITY OF TWENTE.

Master thesis

A SOA-based architecture for dealing with D&D requirements in the homecare domain

Author:	DUC BUI VIET
Date:	August 08, 2011
Graduation committee:	Dr. Marten van Sinderen Department of Computer Science Email: m.j.vansinderen@utwente.nl
	Dr. Maria- Eugenia Iacob Department of Information Systems and Change Management Email: m.e.iacob@utwente.nl
	Alireza Zarghami Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) Email: A.Zarghami@utwente.nl

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands
Telephone: +31 (0)53-489 9111
Fax: +31 (0)53-489 2000
E-mail: info@utwente.nl
Website: http:/www.utwente.nl

UNIVERSITY OF TWENTE.

Preface

Until the moment I write this line, my thesis lasted for 6 months. I started my thesis with a failure. In the first meeting, with uncertainty and a try-and-error mind, I came to my supervisor and proposed my own ideas. He was so nice to give me chances to decide my own fate: spend more time to investigate my direction that even can be extended to be a PhD project or follow his suggestion on another direction. Being afraid of the time limit, I chose the second one: doing the thing that I did not intend to do. I failed to try and failed to take the risk.

However, on the bright side, another door was opened to greet me. I was assigned in the Ucare project, performed a research about Service Oriented Architecture (SOA). Having background in computer science, I got into the topic quite fast and soon discovered the magical capabilities of SOA. More than any generations of software philosophy, SOA bridges business ideas and technology closer. It is SOA that opens new horizons for business man by creating the possibilities to connect, to build and to re-build a system of systems with less time and effort. Based on that idea, the Ucare project, where I used to be a member, wants to develop a new framework integrating existing services in homecare domain. As a compensation for the previous nonsuccess, as mentioned above, I took the most interesting topic in the project, I think, to be my thesis topic: enabling dynamicity and diversity in the homecare domain. The result is quite positive: a SOA-based architecture is created and operationalized. I am satisfied with that.

I am indebted to numerous people; without them, I could never have finished my study on time with an acceptable quality. First, I am particularly grateful to Mr. Van Sinderen for introducing me to the field and giving me the very important and high-level direction of my thesis. Second, I would like to thank to Ms. Maria Iacob for her careful feedbacks and valuable advices for each small section of my thesis. Lastly, there will be never sufficient words to express my obligation to Shahin for his patience, his friendship, his time and his willing to help me.

For all my Vietnamese and international friends, thank you a lot for the unforgettable memories: beer parties, BBQ parties and chit-chat sessions. Being with you, I am aware that I am really far away from my homeland. Being with you, I realize that not only love can a family establish, but also loneliness.

Duc Bui Viet

August 5, 2011

Enschede

Contents

Chapter 1 Introduction

This chapter is dedicated to give an overview about the origin of the host project of the thesis– the Ucare project. An understanding of the root, the obstacles, and the advancement of the host project will help us to have the fundamental background of the domain in both social and technological perspectives. Then, being curious about solutions for the current challenges, we choose one challenge and formulate our motivations based on that challenge. Next, the set of research questions, the research approach and the thesis boundaries will come, steering research activities.

1.1 A need for change in EU healthcare systems

In this part, the high level of business problems leading to the need of improving the healthcare sector will be present. Briefly, as a result of the augmentation of number of elderly people, the current healthcare system is urged to change. European Union and its members chose ITC solutions as one primary way to make healthcare systems adapt better with high demands. U-care project originates from this circumstance.

As the time goes, the 21 century has come. So many problems did the human society successfully overcome; while some is still unsolvable, some new has arisen. The healthcare domain experiences the same phenomenon – the increasing number of elderly people in European countries. Statistical information from European Union's Health Portal shows that, by 2050 in EU "the number of people aged 65 and above is expected to grow by 70% and the number of people aged over 80 by 170%" [1]. This problem, in turn, creates a high pressure on traditional social support systems like healthcare and pension [2] [1]. In addition to the increasing number of elderly people, in 25 EU countries, the total number of healthcare professionals has not increased and tended to go down (Figure 1), weighting the capability of current healthcare systems in satisfying patients' demands.

Having been aware those issues, the European Union already set up a number of policies, programs, and research toward a better health for people in the community. In the field of research, there are also various directions ranging from fighting cancer, ensuring food quality and safety, and combating cardiovascular disease, diabetes and rare diseases. As one of the key research areas, ICTs play an important role on all aspects of healthcare by making healthcare systems more cost-effective, enabling healthcare to be personalized, offering better instruments (medical imaging or supercomputers) and providing channels to access health-related information to everyone [3].

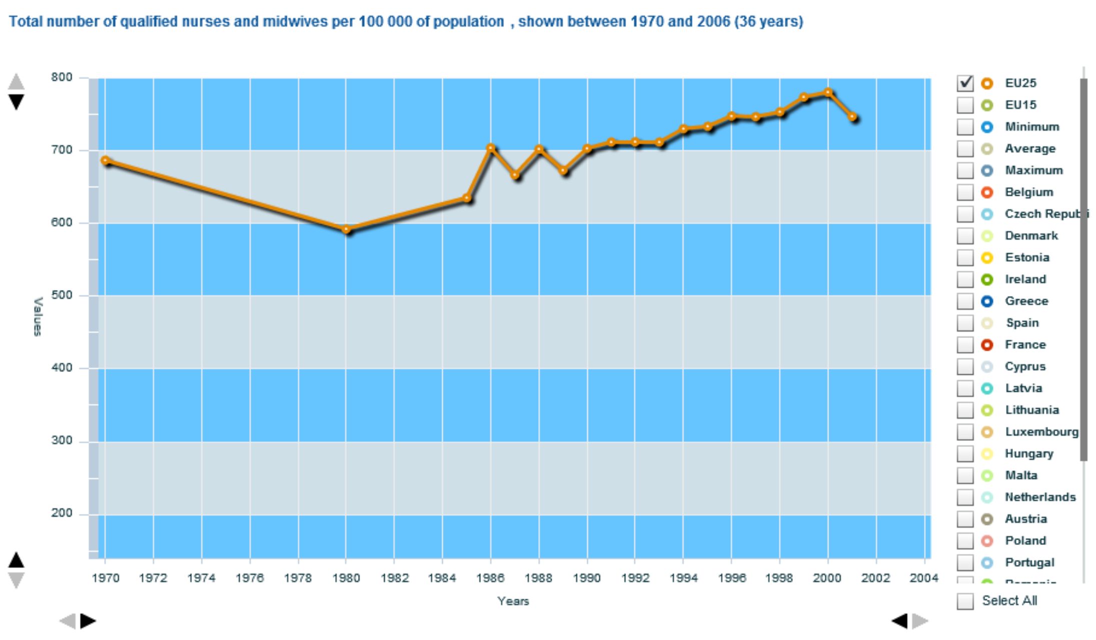


Figure 1: total number of qualified nurses and midwives per 1000 000 of population

However, in a large scale implementation, ITC solutions have their own challenges such as commitment and leadership of health authorities, interoperability of e-health systems, user friendliness, confidentiality and security issues, mobility of patients, etc [4]. In 2004, to surmount those obstacles, an e-Health action plan specifying detailed steps needed to apply e-Health technology is adopted by the EU commission. In this plan, the commission suggests their members to develop their national and regional e-Health strategies to respond to their own specific needs [5].

The Netherlands, in response to the call, already sponsored many research toward ITCs in healthcare; U-care project backed by Ministry of Economic Affairs of the Netherlands is one of them. The next part of the document will present the U-care project's background in more details.

1.2 U-care project

Ucare project is a joint project between CTIT (Centre for Telematics and Information Technology) of the University of Twente, IBM Nederland, Orbis Medisch, Mobihealth BV, TKH Group, IZIT, and CAPE Group.

There are various research directions in the homecare environment like quality of services, privacy and confidentiality of medical data, data management and remote monitoring; however, the problem domain of the U-care project is narrower. This project is designed to "*develop a services layer for integrated home care systems, referred to as the U-Care platform, which provides tailorable, evolvable and non-intrusive care services*"[6]. There are four research themes focused in the project, namely, tailorability, service-oriented architecture, context-awareness, autonomic computing, and E-health and telemedicine services.

One remark is that even there exist many solutions having the characteristic of interoperability and tailorability; this thesis will not compare those solutions to pick SOA and context-awareness approaches. Only reasons to choose SOA and context-aware are addressed.

1.3 Challenges in the homecare domain

Obviously, when integrating various homecare systems into unique one, besides technical issues, the new framework also has to handle social challenges, i.e., relating to human interactions with the new integrated system. The social challenges are focused on dynamicity and diversity while the technical challenges are about interoperability. Those challenges are motivations to SOA adoption which will be discussed right after.

1.3.1 Social challenges

In a homecare system, there are two main stakeholders: care-providers and care-receivers. Care-providers can be professional care-givers from care centers or social care-givers like the neighbors or house mates. Care-receivers are the ones living at the care home [7]. To successfully supply services to these two stakeholders, the new platform has to deal with diversity and dynamicity of continuously changing requirements [8] [9]. *First*, the new platform needs to manage a huge number of care-receivers, and each of them has different behaviors or references/needs, e.g., some prefer to get vibration reminder on PDD instead of voice. *Second*, even for the same care-receiver, his/her preferences are not consistent due to his/her evolution in health conditions. For example, hearing problem of elderly people is normally more and more severe, leading to increasing the volume of alarms or not to use sound devices as alarms. *Third*, with regard to the context of care-receivers, due to the difference in living environments and health problems, there is a diversity of context. For example, context information of care-receivers with cardiopathy is different from context information of care-receivers with brain diseases. *Lastly*, for the same care-receiver, the system always has to deal with changes of context of user. For example, the care-receiver moves from one room to another, causing changes of his/her context in terms of location.

1.3.2 Technological challenges

With regard to technology issues, it is useful to recall that the purpose of the U-care is to build a framework to integrate homecare systems. This properly suggests one of the main problems that system architects have to pay attention to is how to make sure that those current systems can co-operate well for the same goal. Agreeing on that, Klooster et al. [10] also determine that merging home automation, homecare and telemonitoring services is one of challenges for the new framework. More concretely, Eslami, M.Z and M.V. Sinderen [8] articulate that current home healthcare systems are generally stand-alone systems (heterogeneous systems), challenging the framework's architecture to integrate them.

1.4 D&D requirements

Four social challenges mentioned in 1.3.1 can be grouped into two categories: **diversity** and **dynamicity**. Dynamicity is caused by the changes in the context of care-receivers (context dynamicity) and the changes in a care-receiver's preferences/needs (dynamicity of needs/preferences). Diversity in the homecare domain stems from the different needs/preferences of the care-receivers (diversity of preferences/needs) and the different context for each care-receiver (diversity of context).

We name the necessities of the system for dealing with diversity and dynamicity as **D&D requirements** (D&D is the abbreviation of diversity and dynamicity). These requirements, arising in the homecare domain, need to be handled by the Ucare framework.

	User's preferences/needs	**User's context**
Change	Preference/need dynamicity	Context dynamicity
Diversity	Preference/need diversity	Context diversity

Table 1: D&D requirements

1.5 Desired abilities for dealing with D&D requirements

The social and technical challenges raise a set of requirements for the U-care architecture in particular, and for all integrated systems in general. This part of the thesis will elaborate the necessary abilities for U-care architecture.

The D&D requirements, in turn, are believed as motivations to tailorable abilities –**tailorability**- of the system. In details, having tailorability means that the system is able to *provide a set of patient-neutral health-care functions which can be configured and composed according to the needs and references of each individual patient* [8]. Tailorability, therefore, is an important required ability arising from the real problems in homecare. In other words, tailorability assures that the framework can handle the social challenges mentioned in the previous section. In addition, in term of economics, tailorability is also essential because it is economically impossible to build various personalized home-care systems for different individual patients [8].

Concerning the technological challenges, solving problems of combining existing heterogeneous systems means the U-care framework needs to support interoperability because, in definition, **interoperability** is considered "*the ability of two or more systems or components to exchange information and to use the information that has been exchanged*" (IEEE Glossary). In the scope of this thesis, the *interoperability* term is used instead of *system integration* because the relevant problems of system integration can range numerously from political issues, contracts, to technical problems.

Briefly, to reach the goal of providing a services layer for integrated home care systems, tailorability and interoperability are required properties. In the next part, SOA that promises to please these two requirements will be presented.

1.6 SOA adoption in U-care

To achieve tailorability and interoperability, Service Oriented Architecture (SOA) and context-awareness are chosen as solution directions [7]. The decision about context-awareness is based on the idea that context-awareness allows to get context information automatically without disturbing patients' activities, and then change the behaviors of the system according to the context changes. The details of context-awareness will be presented in chapter 3. For the more intricate SOA concept, we will present it concisely so that the readers can have the basic to understand the reasons for SOA adoption and other relevant concepts. In chapter 2, the more profound knowledge about SOA will be examined.

1.6.1 SOA introduction

According to Service Oriented Architecture tenet, software is modularized into independent, well-defined, and self-contained modules [11]. Those modules are called services. The standardized interfaces of services permit communicating then composing different services to support a business mission.

1.6.2 SOA adoption in U-care

As a new way of designing system architecture, Service Oriented Architecture (SOA) is a promising solution that can successfully meet the requirements of tailorability and interoperability. In this section, the ability by which SOA can manage the homecare problems is presented.

With regard to tailorability, SOA solution offers a very flexible way to dealing with D&D requirements. We can simply understand that, in SOA, a scenario of a patient is represented by a workflow which contains connected steps in orders. When there is a change from the patient, instead of changing hardcode as in traditional software, for SOA, the caregiver can easily modify the workflow of a patient by rearranging the order of steps in that workflow.

Perfectly fitting with SOA principles, interoperability requirements are expected to be handled completely. A SOA solution bases on the concept of services. A new framework, instead of working directly with existing systems (legacy systems), will use the standardized services provided by legacy systems. It is the standardization of services of existing systems that overcomes the problem of non-interoperability because standardized services do not depend on operation systems, programming language or vendors. In other words, as long as legacy systems expose their functions as standardized services over Internet environment, the new framework can exploit them. An important remark is that a SOA solution also offers tools enabling legacy systems to export their functions as standardized services.

1.7 Motivations for the research

In the previous parts, based on interoperability and tailorability, the arguments to choose SOA are clarified. However, for U-care project in particular and for the home-care domain in general, there is no in depth research about how to choose and design well SOA-based architecture in combination with other methods to handle tailorability and interoperability. Therefore, needleless to say, a research about SOA-based architecture for the homecare domain is essential. Positive outcomes of research about this topic, in the long-term, can provide a firm reference to develop homecare systems.

1.8 Research objective

The objective of this research is to give a description and an evaluation of an SOA-based architecture for handling the D&D requirement in the homecare domain.

1.9 Problem statement

As one part of the larger U-care project, the problem statement is delivered from the problems of U-care with a narrow scope – that will be described in thesis boundaries.

First, the unavoidable dynamicity of user-context causes many difficulties to homecare systems, leading to frequent failures. Especially, in the homecare domain, due to the very strict requirements in safety and reliability, failures at any level are not acceptable. This requires that homecare systems need to be developed with scrutiny on dynamicity managing capability.

Second, in the homecare domain, there is a wide range of service providers with diverged services like services to provide biosigns (blood-pressure, heart-rate, weight, etc) and context information (location, temperature, etc). Those services, however, are not well connected to deliver greater benefit for neither care-receivers nor care-providers.

1.10 Research question

Main research question: What are the properties of a SOA-based architecture for dealing with D&D requirements in the homecare domain?

In order to answer this research question, the following sub-research questions will be examined.

Sub-research questions:

1. What are the sources of dynamicity and diversity in the homecare domain?

   To enable a SOA-based architecture to deal with D&D requirements, we should know how a care-receiver can cause dynamicity for the system. Having knowledge about that, we can predict the possible stimulus as inputs for the framework.

2. What are the techniques that can handle D&D requirements?

   Answering this question will help us to have an overview about possible ways to overcome trouble caused by dynamicity. As shown in the 3rd chapter, two approaches, namely user-context awareness and services composition, with many their techniques are presented.

3. What are the other domain requirements and constraints that influence architecture design decisions for dealing with D&D requirements?

   Because there are a number of applicable techniques, additional business requirements and domain constrains are taken into account to help us select the most suitable one.

4. How should a SOA-based architecture be designed in order to be able to deal with D&D requirements?

Based on selected approaches, a SOA-based architecture will be proposed. This architecture, then, will be implemented as a prototype.

5. To what extent does the proposed SOA-based architecture satisfy D&D requirements?

   Concerning the main purpose of dealing with dynamicity and diversity, the architecture will be evaluated in terms of the supported flexibilities.

1.11 The thesis boundaries

The previous parts introduces aspects of the U-care project including its motivations, its challenges, its scope (and assumption), its properties and expected results. Due to the time being, it is not realistic to cover all problems of the U-care project in this thesis. Therefore, in this part, the scope of the thesis, in which targeted problems and assumptions, will be explicitly shown.

1.11.1 Scope

For the entire project, as mentioned above, U-care opts to deal with interoperation and tailorability. However, **this thesis is merely focused on tailorability**. Tailorability is expected property to handle D&D requirements.

We exclude the **context diversity** from our scope.

Concerning technical instruments, the **Enterprise Service Bus (ESB)** as mediation architecture pattern will be used.

The targeted context information of dynamicity is run-time changes from care-receivers, including location, time and status.

Other health-related information like whether the medicine is taken is also considered.

Collecting and analyzing the context information (at low abstraction level) are not in the scope.

The proposed architecture will be evaluated in terms of flexibility.

1.11.2 Assumptions

The focus of this thesis is homecare domain (a sub-set of healthcare domain).

When mentioning architecture, the thesis refers to software architecture.

The care-givers' actions are considered accurate and on-time.

The system manages to work with different types of devices properly. In other words, we exclude the dynamicity and diversity related to hardware devices.

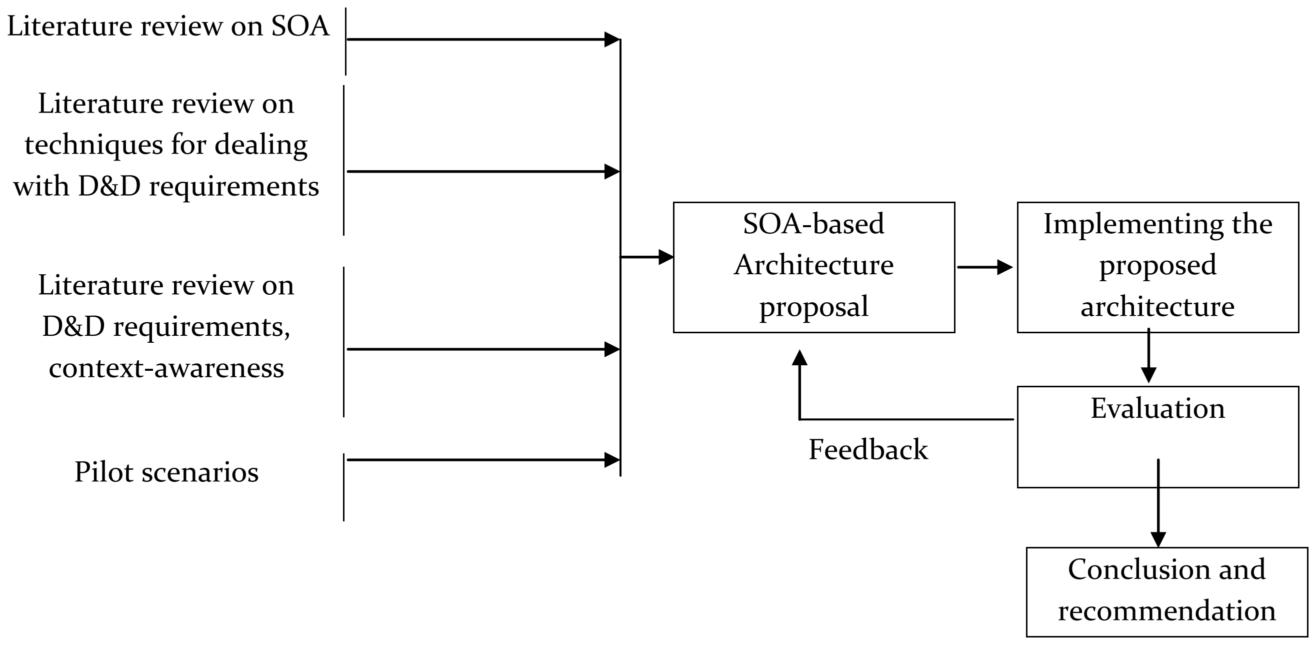
All context information is well collected, analyzed and provided by the third parties.

The infrastructure of the system works properly with no crashes or failures.

The infrastructure is robust enough to handle huge amount of data.

1.12 Research approach

The process of conducting the research is depicted in the following graph.


Literature review on SOA
Literature review on techniques for dealing with D&D requirements
Literature review on D&D requirements, context-awareness
Pilot scenarios
SOA-based Architecture proposal
Implementing the proposed architecture
Evaluation
Feedback
Conclusion and recommendation

Concerning the theoretical content, we will examine the literature to find the techniques for dealing with dynamicity and diversity. Those techniques will be investigated to see how each of them can fulfill the requirements arising from pre-defined scenarios. Then, a SOA-based architecture will be proposed.

In the practical part, after the implementation of the proposed architecture on a process engine (WebSphere Lombardi), its quality will be evaluated in terms of flexibility.

1.13 Report structure

The remainder of this report is structured in the following chapters:

Service Oriented Architecture – chapter 2 – gives a short introduction to Service Oriented Architecture including fundamental concepts like services, service compositions, Enterprise Service Bus, providing the background of many techniques in chapter 3.

Approaches for handling D&D requirements- chapter 3 – presents two approaches, namely, user-context awareness and SOA compositions that are exploited to handle D&D requirements. This chapter answers the sub-question 2.

In order to select the most suitable architecture and technique, chapter 4, titled "*Additional requirements in the homecare domain*", discusses the influential and additional requirements and constrains like safety criteria, stakeholder ability. This chapter answers the sub-question 1 and 5.

In chapter 5, method selection and the proposed architecture will be specified. Based on chapter 3 and chapter 4, one (or more) composition method(s) will be selected. Then, inspired by reference architectures of SOA and use-context awareness and the selected composition method(s), we describe the design steps resulting in an architecture design. This chapter, obviously, is the answer to the sub-question 4.

Chapter 6 – *Implementation of the proposed architecture*- discusses the way to develop a prototype of the proposed architecture. The reminder scenario will be used to illustrate the operation of the architecture. This chapter is also answer to the sub-question 4.

Chapter 7 is dedicated to the *evaluation of the purposed architecture*. This chapter can be considered as the end of the answer to the sub-question 5.

Finally, chapter 8 closes this report by discussing the potentials and drawbacks of the proposed architecture in the homecare domain. The recommendation for the future architecture of the Ucare project will be given lastly.

1.14 Summary

This chapter covers the following issues:

- The increasing number of elderly people and the declining trend in the number of healthcare professionals burden the current healthcare support systems.

- Ucare project aims to develop a platform for integrated homecare systems.

- Ucare project adopts Service Oriented Architecture and context-awareness approaches.

- We define the concept of D&D requirements.

- This thesis is focused on handle the D&D requirements thus supporting tailorability by developing a SOA architecture design.

Chapter 2 Service oriented architecture (SOA)

The Software industry has witnessed a very rapid evolution of enterprise architecture designs that software engineers apply to develop software systems. Looking back at the history, we can observe the evolution of enterprise architecture from the oldest single-tier client-server architecture, two-tier client-server architecture, distributed Internet architecture, hybrid Web service architecture to the most advanced one - Service Oriented Architecture [12]. This chapter will clarify the definition of SOA and its key concepts like services, web services, service composition, Enterprise Service Bus (ESB). Then, criteria to evaluate the quality of SOA-base architectures will be presented. To have a clear view about the proposed architecture, each concept is concretized in the context of the homecare domain and our proposed architecture.

2.1 Definitions

From the beginning of this report, the "*SOA-based architecture*" term is repeated more than one time. In this report, this term refers to an architecture style that employs and supports service-oriented components. In other words, a SOA-based architecture is the architecture of SOA-based applications. To profoundly grasp the insight of this term, it is essential to understand Service Oriented Architecture.

In literature, there are many definitions of SOA. However, in this report, the one from a project of The Open Work Group is chosen because this group includes top technology companies such as HP, IBM, Capgemini, Oracle, and SAP; therefore, it is apparent that this definition is accepted widely. Service Oriented architecture is defined as follows:

> *"Service-Oriented Architecture (SOA) is an architectural style that supports service orientation." [13]*

Then, they also explain the concept of service orientation as *"a way of thinking in terms of services and service-based development and the outcomes of services*" and an architecture style as "*the combination of distinctive features in which architecture is performed or expressed*" [13].

From this definition, it appears many important concepts concerning a SOA-based architecture, for example, the concepts of services, service-based development. Understanding these concepts in SOA context is crucial to understand and then to design a SOA-based architecture.

2.2 Key concepts

As mentioned in the previous part, to understand a SOA-based architecture in depth, we need to examine the fundamental and related concepts in SOA.

2.2.1 Services

Service orientation stems from a software engineering theory known as "*separation of concerns*" [12]. The main idea of this theory is to divide a large problem in to a series of individual concerns, allowing the logic to solve a problem to be decomposed into smaller pieces. This theory was applied widely; for example, in the object-oriented programming approach, those decomposed pieces are objects and classes that can deal with individual concerns of a large problem [12].

In service orientation, similarly, a service corresponds to a small piece of a large logic that can solve/handle the entire problem. This way of analyzing service orientation matches with our preceding definition that considers service orientation as way of "*thinking in term of services*". The Open Work Group also describes a service in more details as "*logical representation of a repeatable business activity that has a specified outcome*".

In SOA, based on the abstraction levels, it is possible to classify services into three types of services, namely, application services, business services and process services[12]. These three types of services, in turn, establish three layers of a SOA application and will be described latter.

-Application services

Application services aim to exploit and reuse the functions from new or legacy application environment[12]. Therefore, instead of working directly with technology-specific functionalities, at a higher level, business services will use services provided by application services. In SOA-based architectures, application services belong to the lowest level (figure 4).

Take the homecare domain for example; we can see many application services such as services to get the blood pressure, to get the temperature. Those are application services because they process raw data from technological devices, i.e., sensors, to extract information needed to business services. Another well-known application service is the communication service that connects the hardware devices (TVs, tablets, and lamps) to software systems in hospitals. This kind of service allows the software system to work with hardware devices without knowing technology-specific details.

-Business services

Business services represent business logic units. A business service can be considered as "controller to compose available application services to execute their business logic" [12]. Business services can be categorized as task-centric business services that present task or business process and entity-centric business services that encapsulate specific business entities [12].

In the reminder scenario (Appendix A), with the purpose to send a message to remind the care-receiver to take medicine, a business service, so called "sending reminder", is the composition of application services like services to convert a remind content to device-targeted format, communication services and sending messages services. To get care-receivers' information, one business service, so called "requiring user-context" service, is built up by various application services to get information about heart rate, weight, blood pressure.

-Process services

Process services involve other business services to execute a business process according to predefined workflow logic[12]. In the reminder scenario, a business process will be developed to take charge of the whole procedure to remind the care-receiver to take medicine. This business service is a process service because by executing this service, a predefined process is deployed.

Concerning the service level, services provided by the U-care framework are in process services (figure 4) because the framework integrates existing services into pre-defined service building blocks that can be reused in wellness and care applications [6].

2.2.2 SOA parties

Previously, three types of services concerning SOA are discussed. In this part, different roles of parties participating in SOA applications will be introduced. Then, we will position the role of the architecture we intend to develop.

There are four main roles, namely, service providers, service requestors, service aggregators and service brokers[11-12]. *Service providers* are organizations (or individuals) who develop and provide services that obey open protocols and standards[12, 14]. *Service requestors* have abilities to discover desired services, to generate request messages and to compose a application from available services [12, 14]. *Service aggregators* (intermediaries) have a dual role - the role of service requesters and the role of service providers. As service providers, *service aggregators* provide composite and higher-level services to service clients; as a service requestors, service aggregators acquire services from other service providers [11]. *Service brokers*, a special type of service providers, offer additional services such as registry services, security and authorization services and supply additional information about reliability, quality, and trust-worthiness of services. The relationship between these parties is illustrated in the following figures [11].

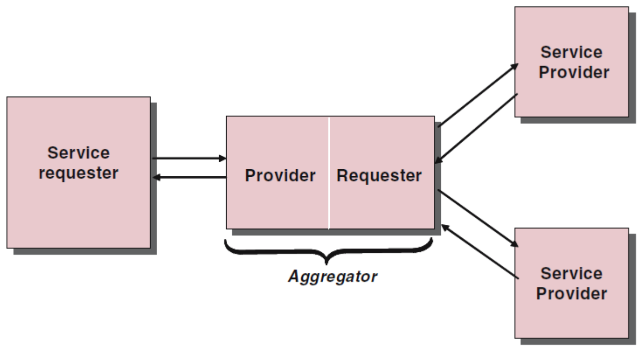


Figure 2: Service aggregator

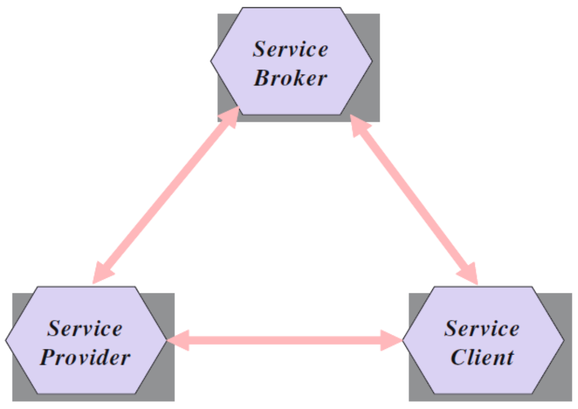


Figure 3: Service broker

To determine the role(s) of the proposed architecture, it is necessary to recall existing services and the functionalities of the proposed architecture. Regarding existing services, many legacy systems provide care-receivers' data like activity level, heart rate and blood pressure; these systems play the service provider's role. The SOA-based framework, on one hand, composes the services offered by legacy systems. On the other hand, the services provided by the U-care framework, in turn, are used by other organizations, e.g., a reminder service (appendix A), are probably used as a service block in a more complex business processes of a healthcare management system. As a result, the U-care framework is an aggregator.

2.2.3 SOA-based architectures in layers

In the previous section, three services types – application services, business services and process services- are examined. These three concepts constitute three adjacent layers of SOA architecture. The whole SOA architecture in layers is depicted in the following figure.

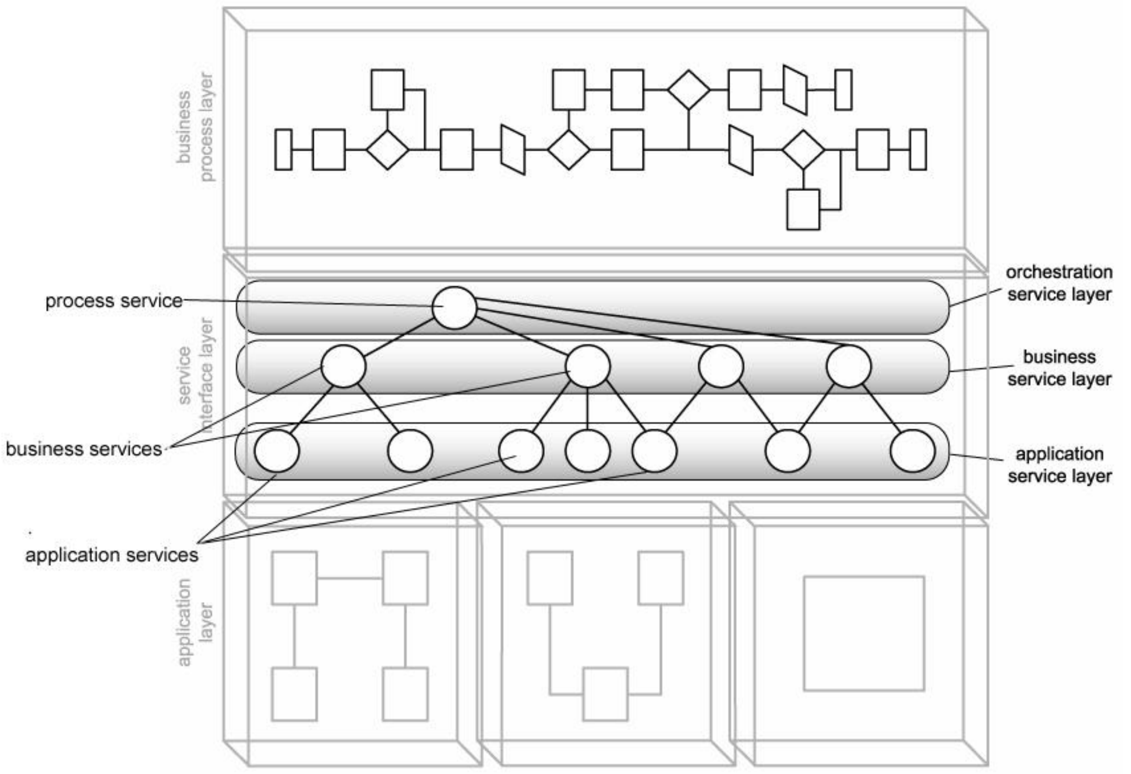


Figure 4: The three primary service layers. [12]

This picture expresses the mechanism for coordinating services of different layers. The lowest layer, the application layer contains the legacy systems that provide technology-specific functionalities that are handled by the application service sub-layer in the service interface layer. The business service sub-layer, one upper level of the application service sub-layer, consumes various application services of the application service sub-layer to establish business services.

2.2.4 Service composition
There exists a principle that a service layer collects services provided by lower abstraction levels and composes them into one composite service. For example, process services (at orchestration service layer) use business services (at business service layer). Therefore, logically, it is vital to have a way to describe the orders of services, to transmit the data from one service to another, or to specify the condition to perform an activity, i.e., a service. In SOA realm, those responsibilities are taken charge by a composition language like Web Service Business Process Execution Language (WS-BPEL). The OASIS (Organization for the Advancement of Structured Information Standards) consortium defines WS- BPEL as "*a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners*" [15]. In other words, the composer will utilize WS-BPEL to model the behaviors of business processes.

2.3 Enterprise Service Bus (ESB)
In reality, services are provided by different services providers with different technologies. There are two fundamental ways to handle those technological and semantic mismatches. First, each service provider has to develop an interface for each connection with other providers, resulting in a point-to-point topology. For SOA, this approach is not adopted because of its limited scalability and its complexity. The second approach applies hub-and-spoke integration patterns. In this approach, Enterprise Service Bus (ESB) is introduced as an integration layer that permits loose coupling and can separate the integration logic into manageable pieces [11].

Enterprise Service Bus is defined as "open, standards-based message bus designed to enable the implementation, deployment, and management of SOA-based solutions with a focus on assembling, deploying, and managing distributed SOA" [11]. With regard to services and service composition, ESB plays a crucial role because it works as a connectivity layer between services by controlling message flows, routing and sending messages [16]. Besides that, ESB is responsible to support security, policy, reliability, remote configuration, and accounting [17].

2.4 Quality criteria for SOA-based architecture
So far, this chapter introduces some fundamental concepts of SOA. However, because the ultimate purpose of this thesis is to develop a SOA-based architecture, quality criteria of SOA-based architectures need to be examined. This part gives a basic view about quality attributes of general SOA-based architecture.

There are a number of quality attributes in the context of SOA. Starting from typical business goals of an organization for its system (being agile, being first to market, enabling easy and flexible integration with legacy system), a set of quality attributes in the context of an SOA are determined. The SOA approach brings positive influences on some attributes while not supporting others attributes properly. The long list of attributes includes interoperability, reliability, availability, usability, security, performance, scalability, extensibility, adaptability, testability, auditability, operability, deployability, and modificability [18]. The detailed descriptions of these quality attributes and the extent of influence of SOA are given in appendix B.

To design the entire architecture of the homecare domain, each of those quality attributes has to be considered carefully. However, to reach the final decision, many tradeoffs among the quality attribute requirements have to be made so that the SOA architecture can meet the business goals [18]. Therefore, the preceding list will be shortened with regard to the specific requirements of homecare domain. The way to evaluate the SOA-based architecture With regard to its abilities to deal with D&D requirements will be presented in chapter 7.

2.5 Summary

In this chapter, we present the following:

- Definitions of Service Oriented Architecture
- Concepts of services including application services, business services, and process services.
- Parties participating in SOA-based architecture
- Service composition
- Enterprise Service Bus
- Some quality attributes to evaluate a SOA-based architecture.

Chapter 3 Approaches for handling D&D requirements

This chapter describes two approaches, namely, user-context awareness, web service compositions to the purpose of handling D&D requirement. The chapter is structured as follows: first, the concept of user-context awareness and how it relates to our problem, i.e., D&D requirements, will be introduced. Next, some feature architectures for user context awareness will be described. Third, a literature review will be conducted toward composition methods in SOA to handle dynamicity and diversity.

3.1 Context, user context and user-context awareness

In this thesis, the definition from Dey et al [19] is chosen because it gives us a simple and accurate view toward context to apply in the homecare domain.

> "Context: Any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects"

As stated in this definition, there are three types of context entities, namely, places, people and things. User context, in this thesis, refers to people entities that can be either individuals or groups, co-located or distributed [19]. **Regarding user context**, four attributes of context information are determined: identity, location, status (activity), and time [19]. **Identity** is a unique identifier to an entity. **Location** can include position, orientation, elevation, co-location, proximity or containment. **Status**, for a person, can be physiological factors (vital signs, tiredness) or current activity (reading, talking). **Time** characterizes a situation by providing historical attributes such as a timestamp, or a time span [19].

Respecting the home-care domain and with the focus on the user context, the example of each preceding concept is given as follows. A care-receiver can be assigned an Identity as a unique number that allows the system to distinguish two care-receivers. Location information of a care-receiver from sensors shows where the care-receiver is, for example, "kitchen" or "bedroom". The most common statuses of a care-receiver are body temperature, heart rate, blood pressure and respiratory rate. The value of time attribute, provided by timers, specifies the moment that a care-receiver performs an action.

Context awareness, context-awareness computing or ubiquitous computing can be defined as "**application's ability to adapt to changing circumstances and respond according to the context of use**" [20]. User context awareness, therefore, can be considered application's ability to adapt and respond to context changes of users.

3.2 User-context awareness and D&D requirements

The two concepts – D&D requirements and user-context awareness - have a tight relationship. D&D requirements have three elements: dynamicity of context, dynamicity of preferences/needs and diversity of preferences/needs. *First*, dynamicity of user-context refers to the changes coming from care-receivers; therefore, to handle user-context dynamicity, we need a method that takes the information of the context changes (identity, location, status and time) as inputs and, then, directs the system's behaviors in accordance with these changes. This ability of the system matches exactly with the purposes of the context awareness defined in 3.1 – adapting to changing circumstances and responding accordingly. In other words, we can use user-context awareness as one feature of systems to support user-context dynamicity. *Second*, With regard to users' preferences/needs, there are some preferences that relate to users' context, e.g., not using lights as alarms after midnight (time attribute in this case). However, there are some preferences independent of context, e.g., Jan prefers vibration alarms to sound alarms. Therefore, user-context awareness is able to dealing with preference diversity and dynamicity affected by user context while having limitation with independent preferences/needs.

3.3 Architectures for user-context awareness

User-context awareness covers enormous topics ranging from analyzing raw data to get useful information, extracting and reasoning to get high abstraction levels of information, discovering suitable services, to changing business processes in according with environmental changes. However, in this thesis, there are two important assumptions: first, many third parties are providing context information at high level of abstraction; second, we target to use SOA composition methods to deal with dynamicity. As a result, in this part, mere the architectures of user-context awareness systems, which give the ideas about compulsory parts, are considered.

This part presents two prominent conceptual frameworks for handling context. These frameworks will be used as references inspiring the proposed architecture to make it possible to support dynamicity through context awareness.

3.3.1 The Context ToolKit architecture

Dey, et al., [19] introduce a conceptual framework that can transform and collect contextual information. This framework is composed of five components, namely, context widgets, interpreters, aggregators, services, and discoverers. From the application perspective, the functions of each component are described briefly as follows:

Context widgets "encapsulate context information and provide methods to access it"[19]. Context widgets are responsible to *notify* applications about context changes, and can be *queried* or *polled* by applications.

*Interpreter*s "transform context information by raising its level of abstraction" [19]. For example, the interpreter can conclude that there is a meeting if it receives information about many people in a room (from one sensor) and high sound level (from another).

Aggregators collect context information that an application is interested in [19].

Services are components that perform actions on behalf of applications [19]. Turning on the light and sending a message are examples of services.

Discoverers manage information about all components and their capability[19]. Discoverers support applications by providing usable components, their names and their identities.

The interconnection between these components is depicted the in following example:

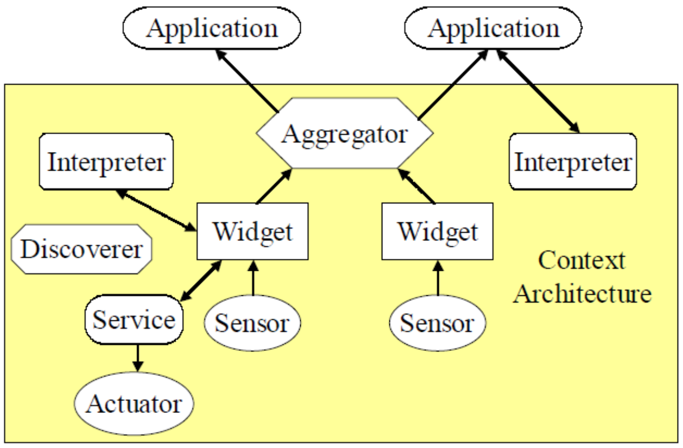


Figure 5: An example of Context Toolkit architecture [19]

This example shows a context-aware architecture style with two sensors, two widgets, one discoverer, one service, two interpreters, one aggregator and two applications. The discoverer holds a registry containing information about all components and their capability. Based on the information from the discoverer, the aggregator can find relevant widgets and interpreters that are interesting for applications. The sensors provide context data to widgets. Widgets store context information and allow interpreters and applications to access context data. Applications can query or are notified by aggregators or from interpreters to get the high level abstraction of data.

3.3.2 Socam (Service-Oriented Context-Aware Middleware) architecture

Suggested by the name, Socam architecture bases on OSGI framework – a lightweight framework for delivering and executing service oriented applications [21]. In addition, OWL (Web Otology Language), that enables context sharing and context reasoning, is applied so that context-aware applications can share and access context information easily.

This architecture has the following components (Figure 6):

Context providers collect context information from external and internal sources, and then process that information to achieve the level of abstraction and finally represent them in OWL.

A context interpreter consists of a context reasoner and a context knowledge base. The context reasoner interprets low-level context to provide high-level context, resolves context conflicts, and maintains knowledge base consistency. The context knowledge base allows other components to add, query, delete or modify context data.

Service-locating service (SLS) provides discovery services including locating context providers, tracking and adapting changes of physical sensors, and enabling context providers to advertise their supported contexts.

Context-aware applications change their behaviors based on the context information. They use the services provided by SLS to locate context providers then to query/listen interested contexts. In accordance with a context change, specific actions will be decided based on business rules that are loaded in the context reasoner.

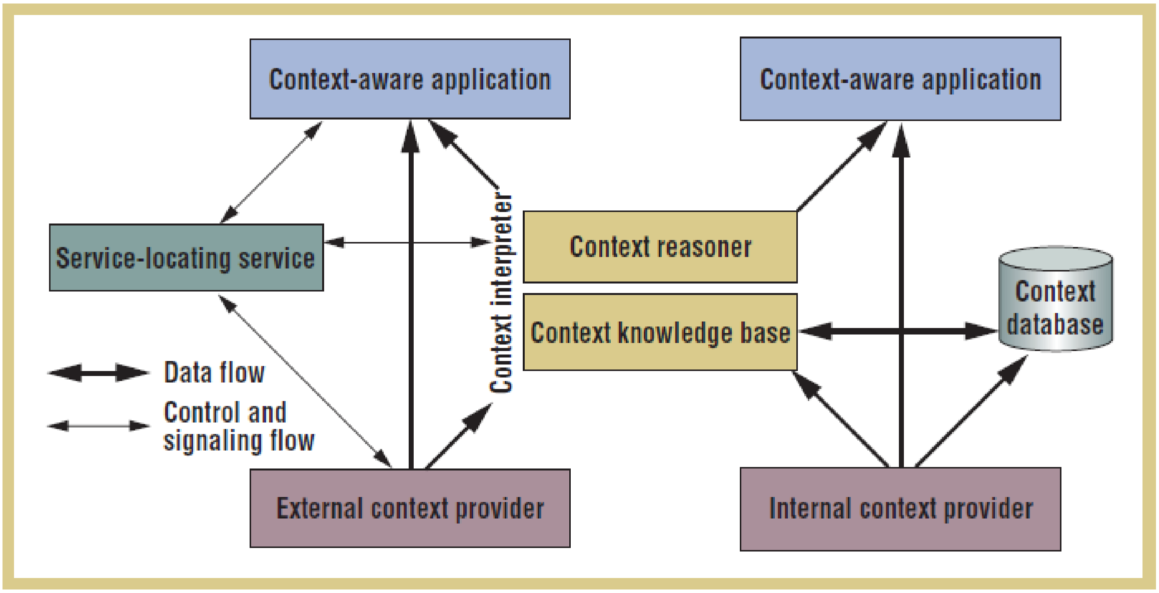

Context-aware application
Context-aware application
Service-locating service
Context reasoner
Context knowledge base
Context database
Context interpreter
Data flow
Control and signaling flow
External context provider
Internal context provider

Figure 6: Socam architecture [21]

3.4 Web service composition methods to deal with D&D requirements

Observing the two reference architectures in the previous parts, we can see that, at the core of those two architectures locate context-aware applications. An application could be considered a reasoner in the sense that, to make decisions from information supplied by aggregators or interpreters, it needs reasoning processes. In this part, one of the approaches to enable the reasoning processes, the web-service composition method, is presented.

3.4.1 Definition and classification

As mentioned in the chapter about SOA architecture, Web service composition methods allow us to compose applications through combining different sets of distributed components [22]. Web service compositions can be classified into static web service compositions and dynamic web service compositions. Static web compositions require developers to compose applications manually and before requested by users; dynamic web service compositions allow composing applications autonomously and on the fly [22].

According to another taxonomy, there are two approaches to web service compositions namely workflow-based composition methods and AI[1]-planning methods [23]. The basic argument of *workflow-based composition* methods is that a composite service is similar to a workflow that involves a collection of services and control and data flow[23]. *AI-planning* methods, on the other hand, articulate that by using logical theorem provers or AI planners, a plan or a process can be generated automatically without knowledge of predefined workflow[23].

It should be noticed that workflow-based compositions are not synonymous with static composition because a workflow-based generation process can be performed *statically* or *dynamically*. For the former one, a predefined abstract process model has to be built before planning compositions, restricting automatic composition to only the selection and binding of web services. *Dynamic workflow composition methods*, however, generate process models and select services automatically [23].

3.4.2 Relations between SOA-based architecture and user-context awareness

A service composition, in other words - a process or a service plan, corresponds to a behavior of application toward an event, or a set of events, in the environment. If an application is able to adapt the order of services in processes (by AI planning methods and dynamic workflow-based methods) or its binding services (by static workflow-based methods) according to the changes of users' context and users' references/needs, we can say that this application has user-context awareness ability in dealing with D&D requirements. Consequently, a SOA-based architecture, which enables web service re-compositions, offers a promising way to deal with D&D requirements.

3.4.3 Service composition methods

Each composition method has its advantages and limitation. This part presents a representative for each composition method so that the features of each one will be revealed. The results of this chapter, combining with the business requirements from the homecare domain, help us to determine an appropriate method for the U-care framework.

Static Workflow-based composition methods

EFlow is a platform supporting the specification, enactment and management of composite services. There are two characteristics of the e-services environment that EFlow aims to handle: a huge number of services and diverse needs of customers. In EFlow, a static workflow generation method is adopted [23]. In detail, a composite service is formed by basic or other composite services and is modeled by a graph containing services, decisions, and event nodes. A service node symbolizes an invocation of basic or composite services. A decision node represents the alternatives and rules controlling the execution flow. Event nodes allow service processes to send and receive [24].

[1] AI stands for Artificial Intelligence.

What makes EFlow adaptive is its ability to discover and bind services dynamically. The composition process can be described in two steps. Since EFlow composition is static, developers have to compose the graph manually[23]. In this step, a service selection rule is defined in each node. In the second step, when the service of a node is invoked, the eFlow engine will call a service broker. After querying services, the service broker will select one appropriate service by applying the service selection rule of that node. The process engine, then, will deploy that service.

Dynamic Workflow-based composition methods

Schuster et al. [25] introduce Polymorphic Process Model (PPM) that supports multi-enterprise processes (MEPs). Multi-enterprise processes refer to workflows consisting of a set of activities provided by different enterprises. Polymorphic Process Model combines both the static and dynamic service compositions [23]. The *static compositions* are enabled by reference process-based MEP. Reference processes-based MEP uses abstract sub-processes which only provide the functionality description and are implemented at runtime by binding Web services. On the other hand, concerning the dynamic part, service compositions are dynamically generated by a state machine that models the possible states of a service and their transitions [23] [25].

AI planning composition methods

Each AI planning method reflects one approach to generate service plans automatically. In details, initializing with a set of possible states of the world (S), an initial state (S_0), a set of possible actions (A) and the preconditions and effects for each execution of each action, an AI planning method will select actions from A and arrange them to establish a service plan that make the world reach the goal state (G) [23].

There are a number of service composition methods based on AI planning. Rao et al. [23] group them into five categories: the situation calculus, the Planning Domain Definition Language, rule-based planning, the theorem proving and the others.

The main characteristics of each method are briefly presented in the following table.

AI planning methods	**Characteristics**
Situation calculus	-Based on the logical language for reasoning about action and change (the first-order language of the situation calculus). -Precondition and effects of web service actions are encoded in the language of the situation calculus. -A deductive machinery uses procedural programming language constructs, e.g., if-then-else or while, services, and constraints to create composite services.
PDDL (Planning Domain	-Exploiting PDDL -a standardized input for state-of-the art

Definition Language)	planners[23]. PDLL is used to describe actions, conditional effects, domain axioms, safety constraints, etc[26].
Rule-based planning	-Generating service plans by using *composability rules* that concern both the syntactic and semantic properties of web services to decide whether two web services are composable. -Syntactic rules are about the rules for operation modes and rules for binding protocols of interacting services. -Semantic rules include rules about message composability, operation semantic composability, qualitative composability, and composition soundness.
Other AI planning methods	-SHOP2, an Hierarchical Task Network (HTN) planner, decomposes tasks into smaller sub-task, until reaching the primary task that can be performed directly[27] -Semi-automatic method uses functionalities and non-functional attributes to select services. If two services satisfy with a requested functionality, non-functional attributes will be used [28].
Theorem proving	-Based on automated deduction and program synthesis.

Table 2: AI planning methods

3.5 Summary

In this chapter, we discussed the following:

- The definitions of context and user-context awareness are given and illustrated by examples in the homecare domain.

- After showing that relationship between user-context awareness can help to handle D&D requirements, we introduce two reference architectures of context awareness.

- Web service composition approachs can be the core-part of context awareness architecture due to its ability to change the order, to insert new services, etc.

- Following by the classification of web service composition methods, we summarize the features of each method to see the advantages and weaknesses which will be considered in chapter 5-selecting one of them to the homecare domain.

Chapter 4 Additional requirements for the homecare domain

The previous chapter summarizes potential methods of services compositions for dealing with D&D requirements. Each method has its own advantages and weaknesses. To select the most suitable one for the SOA-based architecture, additional criteria that can differentiate the utility of each method is needed. In this chapter, in the homecare context, some additional requirements are taken into account to find the desired one.

4.1 Sources of D&D requirements

In our definition of D&D requirements, based on literature, we argue that social challenges cause diversity and dynamicity in terms of user's preferences/needs and context of users (Table 1). To connect the elements of D&D requirements to what happens in the scenarios, this part will present the sources of diversity and dynamicity in the homecare domain. Those sources are initially extracted from two pilot scenarios (Appendix A). Then, we also conduct a literature review about this issue in order to compare with requirements observed in the scenarios thus confirming them.

Scenario	**Context dynamicity**	**Preference/need dynamicity**	**Preference/need diversity**
1	-Jan moves from the sleeping room to the kitchen	- Change the reminders over time according to hearing impairment development	-Jan prefers to take medicine from the closest medicine dispenser
2	- Saturation level drops too low - Care-receivers leave their homes	-Increasing the default volume of reminder voice according to hearing impairment development for John	-Tablet for Marie -Tablet/PDA for John -John prefers vibration reminders

Table 3: Sources of D&D requirements from scenarios

Clearly, from the table, the emergence of dynamicity and diversity proves that D&D requirements are required in the two scenarios. From a different viewpoint, D&D requirements can be classified into three groups as follows:

The first group arises from inconsistent but predictable changes in preferences/needs and context of users, e.g., Jan's moving from the sleeping room to the kitchen is anticipated.

The second group is the changes of health condition which are in the knowledge of care-givers but un-interpretable by the system. Therefore, before giving a decision to this kind of change, consulting care-givers is necessary. For example, the vibration of the blood saturation level is difficult to be analyzed by the system, but it is obviously understandable by care-givers.

The last group is all kinds of exception which refer to users' activities or health condition changes that are unexpected by the system and also problematic for care-givers.

When designing the U-care system for D&D requirements, it is necessary to consider these three groups because the way that the system behaves to each source changes fundamentally. For example, for the first source, inconsistent but predictable changes of users, it is easy to be managed because the system knows what the correspondent reactions have to be done. For the second source, with the interferences of care-givers the system will know the responses. The last one, exceptions, is unknown to the system and care-givers; the system does not how to react (in both static ways and dynamic ways).

This observation from the two scenarios is accordant with literature in the field. Concerning user-related changes, McBryan et al [9] state that needs for changes may be as result of "*what people believe and the way they prefer to or are able to behave and interact with the home care system*".

In sum, there are two different viewpoints about the sources of dynamicity: the first one, based on the characteristic of change, leads to three types of changes; the second one, based on how the change is created, leads to diversity and dynamicity. However, we can map those sources as follows:

How is the change created	**Characteristics of changes**
Reference/need diversity	Group 1 (predefined)
Reference/need dynamicity	Group 1 (predefined), Group 2 (need assistance)
Context dynamicity	Group 1 (predefined), group 2 (need assistance), group 3 (exception)

Table 4: Mapping between two viewpoints of sources of D&D requirements

Here, with regard to diversity and dynamicity of references/needs which are configured by care-givers with profound knowledge about each care-receiver, an exception is hard to happen. The exception is likely to occur during the daily life of care-receivers where a totally strange activity is performed (changes in user-context).

4.2 Additional requirements for the U-care framework

For the U-care Framework, besides supporting D&D requirements, it also has to satisfy various typical requirements of the home-care domain. This part presents supplementary requirements for information systems in the home-care domain. These requirements, then, will influence the architecture design decisions.

Up-to-dateness of context information

When giving health services to care-receivers, the system needs to know the current context of the care-receivers, e.g., location, activity, time. This is important because any changes of user-context can trigger the reactions of the system. Take the reminder scenario (appendix A) for example, if the care-receiver goes to another city, the system could suggest him/her to bring medicines. Consequently, because of the up-to-dateness of context information, it is a must that the system can get updated information all the time by being notified or querying actively.

Safety problems

Healthcare systems are required to be 100% error-free [29-30]. This can be explained that patients' lives relate so closely to the system reliability that any failure of systems can cause severe consequences. The safety requirements of homecare systems can be compared with those of systems in aviation industry[29]. For example, if the system gives an inappropriate decision when the oxygen saturation level in blood drops too low, this decision can cause a loss of life.

Because of the high safety requirement, it is obliged that care-givers have to know exactly the responses of the system toward an event from users. This can be realized by covering all possible users' interactions when designing, monitoring context information in real-time, and having good strategies to dealing with exceptions.

Non-intrusiveness

According to Garde, S. and P. Knaup [29], most of stakeholders in the health care domain, including doctors, nurses, allied health professionals, and administrators are non-technical professional. Therefore, there is a trend to reject IT applications. In order to overcome this obstacle, non-intrusive adaptation is desired. Non-intrusive adaptation means that the system will not ask users for giving opinions or interfering with the system. Every interaction has to be done transparently and without notices to users. For example, also in the reminder scenario, when the care-receiver moves to the kitchen, he/she does not need to specify the location of the reminder that he/she is going to use. This issue, in other words, demands that all interactions between the system and users should be accomplished automatically or at least with minimum efforts of users.

4.3 Summary

In this chapter, the following content is discussed:

- A classification of the changes that cause D&D requirements based on their characteristics.

- Mapping between the manner of the changes and the types of change (which are classified based on their characteristics).

- The business requirements to systems in the home-care domain. The list of requirements includes the up-to-dateness of context information, non-intrusiveness, and safety criteria.

Chapter 5 The proposed architecture

The previous chapters provide sufficient information to design the one SOA-based architecture for D&D requirements. Concretely, we have two reference architectures for user context awareness, a list of potential web service composition methods and business requirements. In this chapter, those inputs are examined with regard to two purposes. First, considering business requirements as criteria, we give the final decision about a method that will be adopted in the proposed architecture. Second, based on reference architectures and the selected technique, we present the design steps resulting in a SOA-based architecture for dealing with D&D requirements.

5.1 Web service composition method selection

From the safety perspective, it can be seen that the static service composition method is more suitable. The reason is that, for the static composition method, care-givers who have knowledge about the care-receiver's health situation can control exactly the reactions and behaviors of the system in case of unknown events. In the other hand, the dynamic service composition, *theoretically*, is able to generate accurately treatment plans if all business rules (constrain rules, action enabler rules, computation rules and inference rules[31]) including safety rules are satisfied. However, *practically*, it is not feasible to generate service plans automatically in all cases with high accuracy due to the highly complex web service environment and the difficulty in capturing behavior in sufficient detail[23, 32]. As a result, having complex clinical knowledge, complex clinical information, and extremely high variability in purposes and interactions[29], the health care domain is too complicated and not suitable to adopt the dynamic service composition method.

With regard to speed to adapt, dynamic service composition methods have more advantages. This is explained by its ability to generate service plans automatically allowing adapting service plans faster thus satisfying the continuous changes in the environment. However, as mentioned above, in very complicated scenarios/environments, the generation of service plans can be a challenge.

About non-intrusive criteria, from care-givers' viewpoint, the static composition method with its limited adaptability needs to inquire information/helps from care-givers more frequently. Nevertheless, the dynamic web service composition method has the ability to reason toward a new event and reproduce a new plan, leading to a high level of non-intrusiveness.

Finally, trade-offs need to be carefully considered. We decided to select the dynamic –workflow composition method. The explanation of this decision is given as follows:

- A service plan is built on a predefined workflow, so care-givers can control the main activities.
- A dynamic composition technique is applied **partly** to the workflow to achieve a completed service plan, so the dynamically generation and non-intrusiveness are obtained to some extent.

In other words, a dynamic-workflow composition method combines the strength of both static and dynamic service composition methods.

5.2 Selection of techniques of the dynamic-workflow composition method

The dynamic-workflow composition method indicates to a family of methods based on the same idea. Therefore, to eliminate the possible confusion between the method and an instance of the method - a concrete way of implementing, hereafter, **we will use the "technique" term to mention to instances of methods**. Therefore, for one method, there are possibly a number of techniques based on ideas of that method. Knowing that the dynamic-workflow composition method will be employed, in this part, we zoom in each of its techniques to find the most suitable one. However, even the dynamic-workflow method has its own difficulties; in the next section, the challenges of the dynamic workflow composition method will be investigated.

5.2.1 Business rules and problems of integrating business rules in BPEL processes

Starting with an abstract workflow, the dynamic-workflow composition method aims to generate completed service plans at runtime. However, the current languages for describing service plans, e.g., WS-BPEL, are not flexible enough to do so. Therefore, all the techniques of the dynamic-workflow composition method attempt to reach the same goal, namely, controlling the combination of business rules and business processes. In this part, after introducing the concept of business rules, those problems of WS-BPEL will be presented.

Business rules

A business rule can be defined as "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business" [33]. An example of business rule is "*if the care-receiver does not take medicine after 30 minutes than the scheduled time, an alarm is sent to care-givers*". Business rules change more frequently than core application functionalities do because they are embedded in business policies which evolve fast [34].

There are four types of business rules[35-36]:

A *constraint* rule is "a statement that expresses an unconditional circumstance that must be true or false"[31]. For example, in the reminder scenario, *a device (TVs, alarms, or speakers) that is nearest the care-receiver is used to send alarm*.

An *action enabler* rule is "a statement that checks conditions and upon finding them true initiates some actions" "[31], e.g., *if the care-receiver take the wrong pills, an alarm is sent.*

A *computation* rule is "a statement that checks a condition and when the result is true, provides an algorithm to calculate the value of a term" [31], for instance, *if the care-receiver goes away from home X kilometer, the reminder process has to be started (X/20) hour(s) earlier*.

An *inference* rule is "a statement that tests conditions and upon finding them true, establishes the truth of a new fact" [31], for example, *if the care-receiver forgets to take medicine 4 times in a week, the care-giver have to go to the care-receiver's home to remind.*

Problems of process-oriented languages

When composing several web services into a service plan, the composer has to deal with business processes and business rules, which have contradicted characteristics. On one hand, business rules reflecting business policies change very fast; on the other hand, business processes are consistent[37-38]. Traditionally, business rules are integrated into one business process thus creating a service plan described by WS-BPEL. For example, in the reminder scenario (appendix A), if we apply the rule "*if the care-receiver's ID is 's1101245x', the dispenser has to be automatically opened*", the reminder process is modeled and simplified like the following graph:

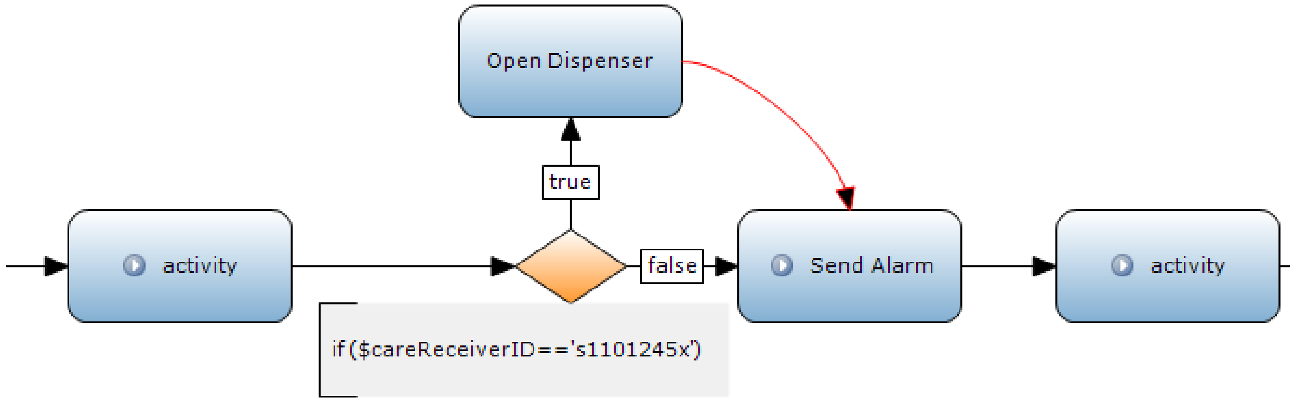


Figure 7 Embedded business rules in busines processes

However, using a process-oriented language, e.g., WS-BPEL, to model those two contradicted entities (business processes and business rules), raises two major problems[31, 34] :

-**Lack of Modularity in Modeling Crosscutting** concerns[31, 34]. For example, in the homecare domain, a business rule "*after invoking any services, the system measures and logs the response time"* is a crosscutting concern because it is scattered in many service blocks of service plans. This makes it difficult to maintain and reuse. For example, due to the policy changes, the preceding rule is modified to "*only after completing the service plan, the system measures and logs the response time*". In order to adjust service plans, the composer has to know exactly where the rule is placed and when it is implemented in the service plan. In a complex service plan, this task is very difficult and time-consuming, resulting in difficulties in maintenance (inflexibility to change).

-**Changing the composition at Runtime** concerns[31, 34]. A service plan, when modeled by BPEL, allows merely dynamic partner binding, and there are no supports to evolutionary and on-the-fly changes. For example, when a web service changes its parameters, there no ways to put this change in effect immediately. Normally, we have to stop the running process, modify and restart. This procedure leads to negative effects such as a loss of customers or unexpected exceptions.

5.2.2 Techniques of the dynamic-workflow composition method

As mentioned above, the ultimate goal of the dynamic-workflow composition method is to build a service plan, based on a predefined abstract workflow and a set of business rules. The following techniques are classified into the dynamic-workflow composition method because they procedure the final service plan in runtime, based on a predefined business process and a set of business rules.

AO4BPEL- Aspect-oriented extension to BPEL

Based on the ideas of aspect-oriented programming, Charfi and Mezini [31] introduce an *aspect* entity. This is an XML file, called aspect file, which has two functions:

- Containing a list of *joint points* which form a *pointcut.* A joint point is a predefined point in the business process.
- At each joint point, an *advice* is specified. An advice is a BPEL activity representing a behavior that should be executed when the associated joint point is reached.

The authors use *pointcut* as the way to separate business rules and business processes, removing the two problems above. An extended BPEL engine (AO4BPEL engine) designed to handle the execution of *pointcut, advices* and *aspects* is also provided.

Service-oriented approach

Rosenberg and Dustdars [38] create a *Rule Interceptor Service* which intercepts all incoming and outgoing Web service calls, map to business rules, and then apply associated business rules. A mapping document is used to map a call to business rules. A Business Rules Broker is introduced to provide a unified access to different rule engines.

Van Eijndhoven's technique

Eijndhoven et al. [37] exploit the power of a business process engine (Aqualogic BPM Studio) and ILOG business rules engine. At the variability points in the process, the process engine sends the request to the rule engine. Based on the input data from the request and the current context, the rule engine evaluates its business rules and returns the result to process engine.

Cibrán's technique

Cibrian and Verheecke [34] also apply the ideas of Aspect Oriented Programming. They use JAsCo, an AOP language tailored for the component based context, to separate business rules and business processes. However, this approach does not use BPEL to describe processes.

5.2.3 Why are business rule engines not suitable?

Some techniques like the ones proposed by Van Eijndhoven [37] or Beer, T., et al.,[39] use an Event Condition Action (ECA) rule engine to fire actions when an interested event occurs and conditions are satisfied. However, we find the two following disadvantages of this approach:

- Since the main purpose of the rule engine is to give decision by applying business rules, the actions invoked by rule engines is simple, e.g., "*when the care-receiver's blood-pressure is too high, send a message to care-givers*". However, for more complex actions containing many steps with transactions between them, rule engines are not powerful enough to compose processes or to transfer messages between steps of process. An example of complicated action can be "*when the care-receiver forgets to take medicine at the first time, the system logs his user-context information into databases and send a notice to care-givers*".

- The above reason leads to one alternative in which a rule engine is integrated with the orchestration engine. However, there are also two concerns in this approach[40]. First, it creates a paradigm mismatch to BPEL programmers, changing their way of thinking to use a rule-based language. Second disadvantage refers to inconsistency for process programmers because they have to work in two service composition environments.

5.2.4 Technique selection

All the techniques listed above belong to the dynamic-workflow composition method. The technique proposed by Van Eijndhoven, which uses both the process engine and the rule engine, is a costly solution because rule engines are expensive. In addition, the cost for those solutions increases because of a high-performance server to host both rule and process engines. Other reasons are emphasized in 5.2.3.

The technique proposed by Cibrán poses two concerns: first, it does not use BPEL –the standard for web service composition[31]- to model the process; second, it is not supported by any process engines.

The technique introduced by Charfi and Mezini is a prominent one in the sense that it merely utilizes one process engine and has a very clear way to separate business rules and business processes. However, this technique has two disadvantages. First, it requires modifying the BPEL engine [31, 37]. Second, in case of managing a set of complicated rules, because of having no supports for rule management, users have difficulties in checking rule consistency, combining rules or solving rule conflicts[31].

Taking into account that Lombardi, a process engine, is available for the Ucare Project and, with regard to the scenarios, the business rule set is not complex, we decide to choose the Aspect-Oriented Approach. In addition, to avoid the modification of Lombardi process engine which can be very intricate, we exploit the power of JavaScript API of Lombardi to support interaction with *aspects* containers and executing *advices*. We will present this approach more precisely in the implementation part.

5.2.5 Why does crosscutting problem matter in our situation?

As the reminder scenario will be used as challenge to the proposed architecture, some may question that why crosscutting problems need to be considered while the scenario description suggests a simple process to send reminder. If crosscutting problems are **not** encountered in this scenario, the reason to method based on Aspect-Oriented Approach[31], which supports crosscutting problems heavily, can be collapsed or, at least, cannot expose its advantages fully. In this section, this issue will be clarified.

Not only in the complicated business processes, are the crosscutting concerns also found in the simple case like reminder scenario. For example, one possible business rule is "*after all steps in the business rules, the context information will be logged into database".* The purpose of this rule is to make possible to audit the reaction of care-receivers. Another example that will be implemented in the implementation part is that "*after the first context inquisition and after receiving new user-context information, the system will match the new location to the endpoint of closest device to the care-receiver*". These two examples present two business rules that scatter in many places of business processes. It is the evidence that even in the simple scenario, crosscutting problems appear.

In addition, for a larger scope- the whole project, when dealing with the real homecare environment, it is very likely that the architecture has to support more complex scenarios. For instance, for care-receivers with Alzheimer disease at early stages, the scenario to guide them to go out for physical exercise can be very complicated. The Aspect-Oriented approach has great potentials for this kind of scenario.

5.2.6 What can an aspect do with regard to D&D requirements?

Above, we mentioned that business rules are materialized in the form of aspect files. However, coming back to our purpose of dealing with D&D requirements, as definition, we have three main sources of dynamicity and diversity: diversity of user's references/needs, dynamicity of or references/needs and user-context dynamicity. Some can raise the question that about the relationship between those three sources and business rules in aspect files. Regardless of the details of an aspect file, in this session, we will clarify what an aspect can do for D&D requirements.

-There are four types of business rules (5.2.1), and not all those rules are used to model the reaction of the system in dealing with dynamicity and diversity. For example, for a constraint rule like "*blood-pressure is a positive value*", it does not model any behavior of the system.

-In 3.2, we contend that user-context awareness is limited in supporting diversity and dynamicity of preferences/needs that are independent of user-context. This disadvantage is overcome by applying business rules that are flexible to describe any preferences/needs. For example, with a preference like "*the care-receiver with ID 'so11566' prefer vibration alarms to sound alarms*", the business rule for it can be described as "*if (ID=so11566) do (select vibration alarms)*". This business rule can be exploited to regulate the behaviors of the system. Briefly, with the ability to recompose business process according to context changes, and the flexibility of business rules to model preference/needs that are independent of context, SOA architectures can deal with all three sources of D&D requirements.

- Business rules modeled in aspects files can be used for all types of dynamicity and diversity sources by different configurations of the condition statements. The following table will show examples of sources of dynamicity and how conditional statements represent them.

Sources of D&D requirements	**Conditional statements in aspect files**
Diversity of preferences/needs	If (ID="s110636") do (action A) If (ID="s122525") do (action B)
Dynamicity of preferences/needs	If (hearing problem level > 8/10) do (action B)
Dynamicity of user context	If (current location="kitchen") do (action C) If (current location="sleeping room") do (action D)

Table 5: Sources of D&D requirements and aspects' condition statements

5.3 The proposed architecture

This part, a SOA-based architecture which is motivated by context-awareness architectures and the selected web-service composition technique (AOP) will be elaborated.

5.3.1 Provided services

Before digging into the design process, we recapitulate services that are in and out of the scope of the U-care framework.

There are many third parties who already master working with healthcare devices, alleviating the system in processing the context information at low-level of abstraction. They provide various services such as location services, biosigns measuring services, and healthcare-device monitoring services.

With regard to scenarios in Appendix A, services controlled by the U-care framework include services to send alarms to PC tablet, PDA or other types of alarms. Even providing services to control alarms (TV, wristwatch and mobile-phone) is not the role of the Ucare framework; however, because there are no providers who support this task, Ucare project will take charges. A calendar service which triggers a business process, in given time, is also provided by Ucare-project.

5.3.2 Design steps

The proposed architecture is inspired by two reference architectures (section 3.3) and the selected web service composition technique (section 5.2.3). The design process has three steps. First, starting at reference architectures, we select the necessary modules that collect user-context information and device information. Second, the modules supporting the dynamic workflow composition technique will be decided. Lastly, we will add some additional modules.

Step 1: Determining modules originated from two reference architectures

The user-context awareness architectures we use as references suggest us the following modules:

1. *User-context aggregator (provided by 3rd parties)*:
   Name in SOCAM: context provider
   Name in Toolkit: aggregator
   Responsibilities:

   -contacting periodically to sensors to update new user-context information (location or blood pressure) users.

   -analyzing the raw data from sensors to get interested information.

Since this module is managed by service providers, it is not in our scope. Therefore, in the proposed architecture, we do not present it. We, instead, show a list of services provided by third parties to illustrate the output of this module.

2. *Device monitor*:
   Name in ToolKit: discoverers
   Name in SOCAM: service-locating service

   Responsibilities:

   -contacting periodically to devices (or directly by web service form 3rd parties) to update new context of devices.

3. *Device facilitator:*
   Name in Toolkit: services and actuators
   Name in SOCAM: service providers

   Responsibilities:

   -executing commands by invoking corresponding devices.

In the context of SOA, the concepts of device monitors and device facilitators are equal to the adaptor concept. **An adaptor** has ability to “provide connectivity, semantic disambiguation and transition services between application and collaboration”[11]. Therefore, an adaptor can not only enable communication in two directions between the devices and the system but also convert the different interfaces, protocols, data format of different devices into the standardized ones for the system and vice versus. As a result, in the proposed architecture, adaptors will be used instead of devices monitors and device facilitators.

4. *Discovery management module*

   Name in SOCAM: service-locating service
   Name in ToolKit: discoverers

Responsibilities:

- searching services in Service Repository based on UDDI technology
- retrieving metadata of services to prioritize services
- selecting the most suitable services based on rule-based or learning-based

Step 2: Determining modules supporting the dynamic workflow composition technique

5. *Aspect database*

   - Storing business rules written in the form of *aspects*. One aspect file contains information of a business rules such as the conditions, the operations and the returns.

6. *Advice repository*

   - Storing a library of predefined *advices* that will be invoked at certain *joint points* in the main process. One advice is a BPEL activity[31].

7. *Process executor*

   -Executing business processes.

8. *Aspect manager*

   -Retrieving *aspect* information, evaluating conditions and determining *advices*.

   -Calculating new values of variables in business processes.

The aspect manager has the functions of a business rule and more. First, as a business rule engine, the aspect manager has the ability to take the variables in the business process as inputs, evaluate and return the output according to the configuration in aspect files (that represent business rules). This chain of activities, in other words, is the execution of one business rule. Furthermore, more than a business rule engine, the aspect manager allows us to insert one sub-process or a service into the business process through invoking advices specified in aspect files. For a simple activity, a business rule engine can be used; however, to more complicated activities with many steps inside, business rule engines are not suitable because it is not specialized to compose and execute business processes (5.2.3). In this sense, aspect managers that insert sub-processes (advices) into the main business processes which are executed by process engines are better solution.

The difference between an aspect manager and a rule engine are threefold. First, a rule engine is not specialized in executing business processes (as actions of business rules) which are possibly invoked by a process. An aspect manager, however, can insert a business process (as actions of business rules) in a process, making new one dynamically and then be executed by a process engine. Second, since rule engines are designed to manage business rules, they support more tasks concerning rules like, check consistency, conflicts between business rules. Finally, an aspect manager can overcome the crosscutting problems.

Step 3: Determining additional modules

However, if the system wants to store historical information for business rules, e.g., "*in a week, if the care-receiver forgets to take medicine, a care-giver has to go to the care-receiver's home to remind*", the preceding modules cannot support. Furthermore, in order to deal with frequent user-context changes, the system needs a mechanism to receive information from *adaptors*. Those reasons stimulate the following module:

9. *Context server*

- Listening to the user-context changes from adaptors.
- Storing context information of users and devices in a database.
- Allowing querying context information by the aspect manager.
- Informing user-context changes to the aspect manager.

The relationship between modules of the architecture is depicted in the following graph.

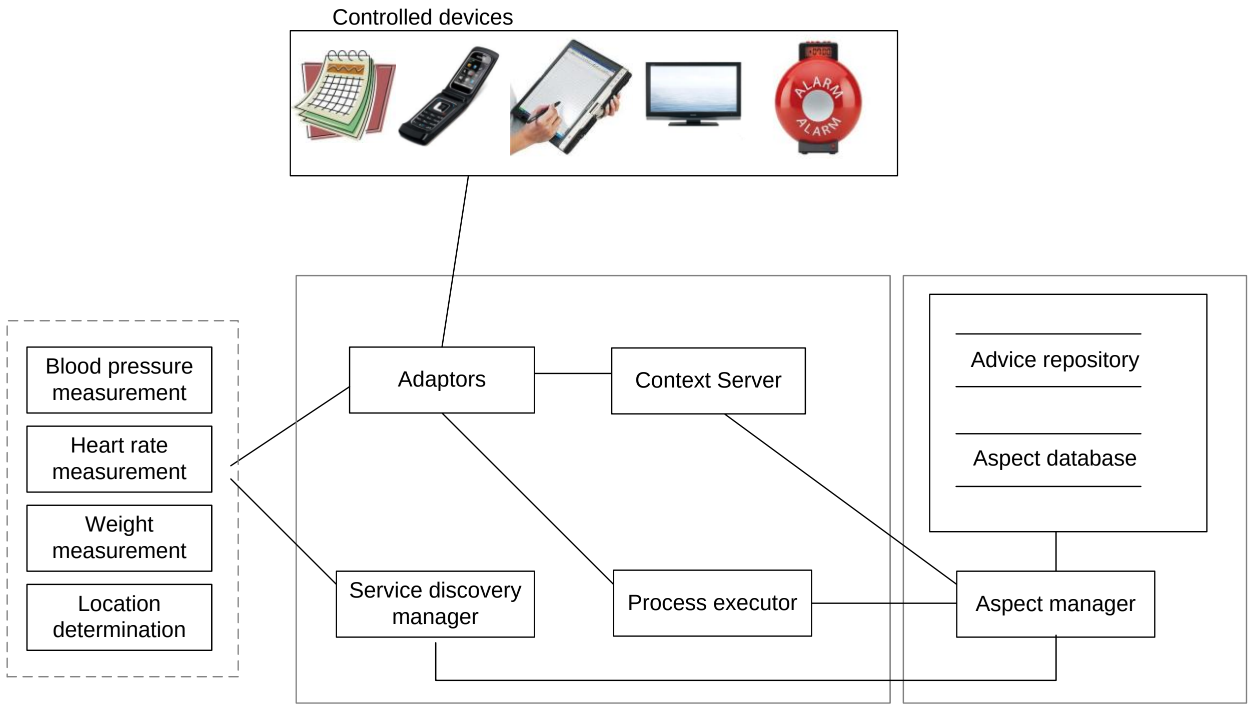


Figure 8: Relationships between modules of the Proposed SOA-based Architecture

The operation of the architecture is described as follows:

Whenever there is a change of biosigns or user-context, *service providers* having the updated users-context information will notify to the system. As a part of the system, the *Context Server* module receives the notification. Next, the *Context Server* performs two tasks: first, updating the change to the *aspect manager*; second, registering the change into *a context database* inside the *context server*. Receiving the context-user information, the *aspect manager* performs calculations based on business rules in aspect files. The results can be new values for parameters in the process which will be update into the main process.

When the *process executor* reaches one *joint point* in the main process, the *aspect manager*, written in JavaScript, is invoked. The main process turns in suspending mode. The *aspect manager* parses xml files that describe business rules as *aspects*. Those xml files are stored in an *aspect database*. By doing so, the *aspect manager* knows that at that joint point, if one condition is satisfied, one *advice* will be performed. The *advice* is modeled by BPEL and kept in an *advice repository*. After finishing executing the advices, the process executor resumes the main process and runs it normally.

The architecture is modeled in ArchiMate language [41] as shown in Figure 9 below. The infrastructure including the controlled devices, DB2 database system and IBM Lombardi locates at the lowest level. Software and devices at the infrastructure provide infrastructure services to send alarms, to provide information of devices like location and availability, to execute business processes, to perform file services (read, delete, and save), and to execute query statements of the database management system. Besides controlled services provided by the infrastructure of Ucare, many external services provided by third parties can be utilized. For instance, there are services to provide blood pressure, heart beat rate, and activity level (biosigns), or location; other services are to control device at home like opening dispensers, or to provide information about whether the care-receiver takes medicines at the dispenser or not. At the upper level situate application components those are described above.

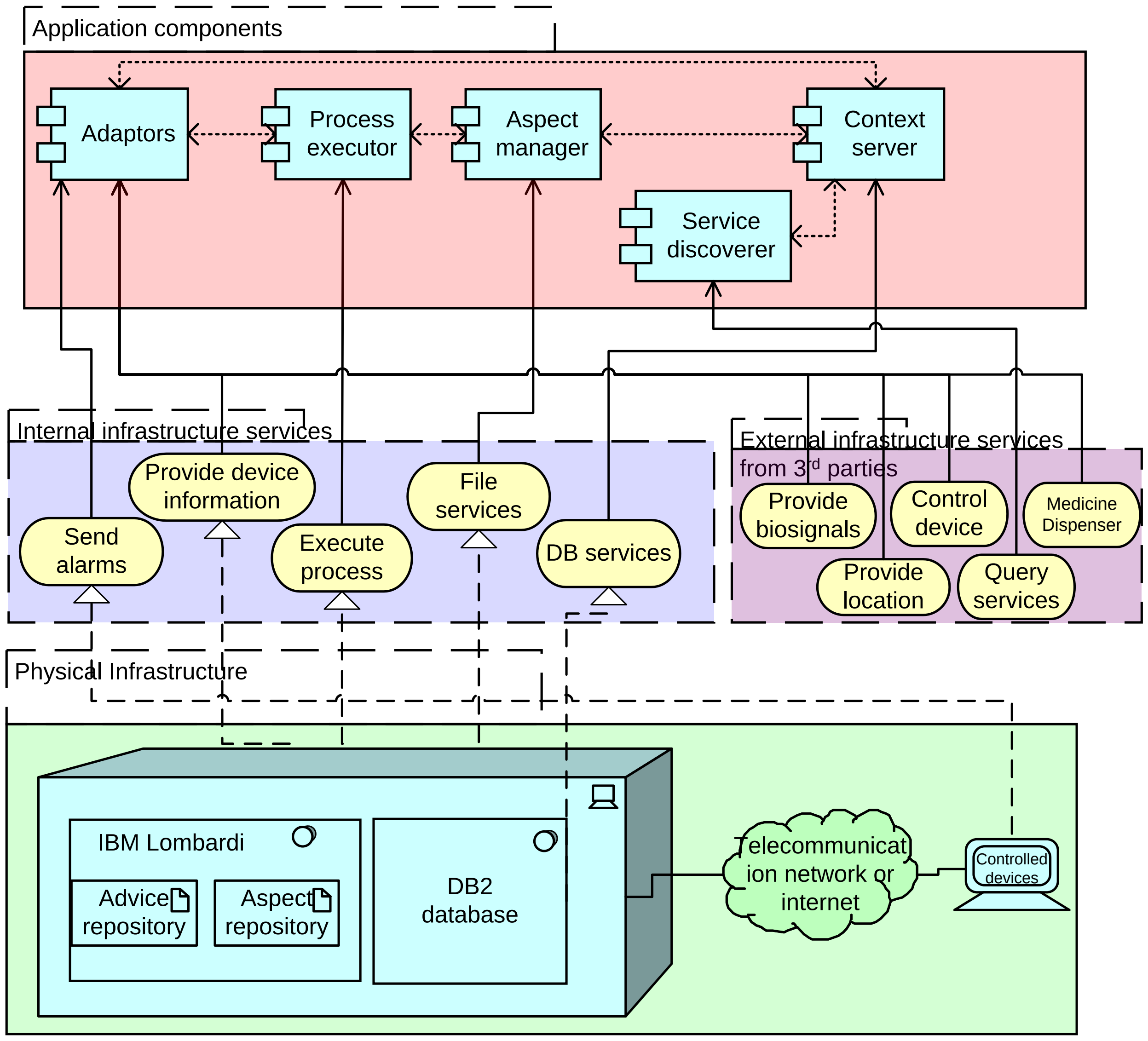


Figure 9 : The propose architecture in ArchiMate language

5.4 Summary

In this chapter, we discuss the following:

- Based on the business requirements in chapter 4, we select the dynamic-workflow service composition method. We also discuss about the two issues of this method.
- The Aspect-Oriented Approach proposed by Charfi and Mezini [31] is selected as an technique of the dynamic workflow composition method.
- Based on the two reference architecture for context-awareness and the selected technique, we propose an SOA-based architecture.

Chapter 6 Implementation

In the previous chapter, the proposed architecture shows necessary components in the system and the relationship between them. In this chapter, that architecture will be implemented so that we can see that feasibility of developing it with the current technology.

This chapter is structured in five main parts. First, we introduce one scenario as a use-case to challenge the architecture. Second, based on the scenario, a list of required services for the scenario is given. Third, the tools and their capabilities applied to implement different parts of the architecture will be presented briefly. Then, concerning the chosen scenario, architecture components and mock-up services will be modeled and implemented. After this step, we have a prototype of Ucare system that has the ability to deal with D&D requirement, which will be assessed at the next chapter.

6.1 The scenario

6.1.1 Scenario description

> *"Jan is an elderly who lives inside an apartment which equipped with homecare applications on top of our application platform. He should intake his medicine at 11:50 PM. The application should remind him to intake medicine several times up to 15 minutes later than the scheduled time. If not taken, the alarm should be sent to the care center. He has hearing impairment developing over the time. He also uses wheelchair, so the doors inside the apartment should be opened automatically. He only can speak Dutch and prefers to take medicine from the closet medicine dispenser (MD) at night. Two MDs, filled with the required medicine, have been installed, one in the kitchen and the other one in the corridor. The MD inside the kitchen has embedded light. TV inside the sleeping room, all the lights of the apartment and a wristwatch can be used as reminder devices for taking medicine. He prefers not to be reminded by lights after midnight. Nancy, as a care-giver, wants to create a service plan. Because she knows his requirement and situation better than IT specialists, she is going to make a service plan in order to assist Jan and remind him to take his medicine on time. She is going to make the service plan based on Jan's requirements, abilities and preferences."*

6.1.2 Additional business rules for the scenario

The scenario above is not complicated, containing no business rules that can lead to major changes in business processes. For example, if the care-receiver moves from room where the TV is used as alarm to another room where the lights are alarms, the service plan is still the same. In this case, the only change is the endpoint (that specifies which device is invoked). Therefore, to demonstrate the ability of handling also the business rules that, when applied, require additional services thus changing business processes, we introduce the following business rule:

-**R1**: *if the care-receiver' ID is "s101hyk", the medicine dispenser will be opened automatically before sending reminder to the care-receiver.*

This business rule cannot be modeled as a fix part in business processes because it is not applicable for all care-receivers. In addition, if the architecture can handle this rule, it has the ability to deal with diversity of preferences/needs because this business rule represents a user's preference.

6. 1.3 Services required by the scenario

Services	**Explanation**
Inform user context information	Whenever there is a change from care-receivers, by mobile sensors, the third parties such as MobileHealth can monitor the change and inform to the system.
Inquire user context information	There are some moments when the system needs user-context information, e.g., when a process is triggered, at the first time, it needs to inquire user context information actively to initialize its variables. This service is given by third parties.
Remind a care-receiver to take medicine	Controlled by the Ucare system, this service is to send command signals to reminder devices like TV, lights, and wristwatch in order to catch the attention of care-receivers so that his/she takes medicine.
Send alarms to care-givers	Controlled by the Ucare system, this service is to send command signals to care-givers in hospitals in emergency situations.
Trigger the process	Ucare system allows composing and executing service plans; however, another system of a clinic, for example, has responsibilities to control those service plans. Therefore, a service to trigger the reminder process from a clinic is required.
Open a dispenser	For business rule R1, this service provided by third parties is to open a medicine dispenser.
Check medicine taken	This service provided by third parties is to check whether the care-receiver take medicine or not.

Table 6 Services in the reminder scenario

6.2 Implementation tools

In this part, the implementation tools which make the prototype of the architecture and third parties possible will be presented.

Mendix is a model-driven platform enabling us to design, deploy, integrate and share business applications[42]. Because of this tool's simplicity to build web services, we use it to develop mock-up Web services, for example, services to send alarms, to remind, and to open a dispenser.

IBM-Websphere Lombardi is a business process manager that allows creating process models, implementing process steps, running and inspecting processes, optimizing and installing process applications[43]. We use this process engine to compose the services created by **Mendix**. In addition, this tool has a very powerful JavaScrip API (Application programming interface), allowing us to develop the *aspect manager* of the proposed architecture.

SoapUI is a testing tool to test various standard protocols and technologies like WDSL, SOAP, and JMS. This software helps us to create the triggering services and services to update user-context information.

6.3 Building modules of the architecture

In this part, the seven modules of the proposed architecture will be developed with the implementation tools above. Although this scenario with its process and rules does not require the supports of all modules, e.g., context database, however, for a general purpose, an introduction of implementing all those modules is necessary. Before presenting the way to build, we will give a short summary for each module.

6.3.1 Context server

As mentioned above, the context server has two main functions: listening to changes informed by third parties and updating changes to the aspect manager. The listening task of context server, realized by Lombardi, is implemented as a service.

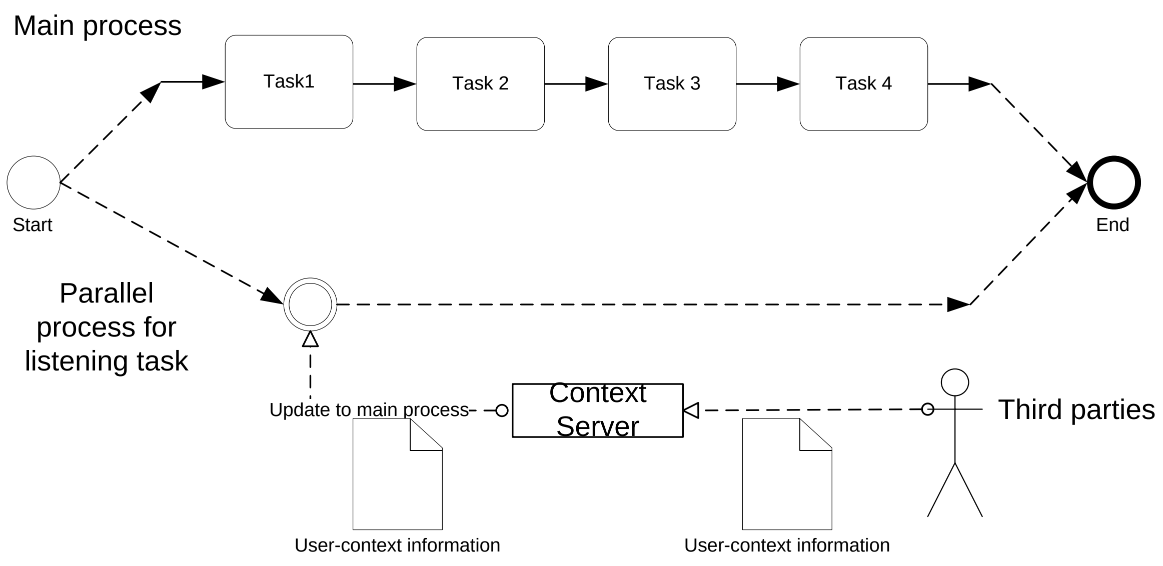


Figure 10 : Context Server

First, this service is exposed as a web service to receive user context information from third parties. By this way, as long as third parties have WDSL of the web service of the listening task, they can notify about changes by calling that web service. Second, in order to update the changes in user context from third parties without interrupting the main process, the listening task is integrated in a parallel process of the main process.

6.3.2 Aspect manager

The *aspect manager* performs three steps: extracting aspect information from aspect files; evaluating aspect conditions based on the values of variables in main processes; and executing the advices (processes or services) when the conditions are satisfied. All these tasks are programmed and placed in a JavaScript files executed by JavaScript Engine of Lombardi. To make these steps clearer, the following example shows the content of an aspect file, and what the *aspect manager* does.

Retrieving the aspect information from aspect files

```
<aspect name="UcareServiceInvocations"
xmlns:cns="http://localhost:8080/axis/services/CounterService"

xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

<partners>
 <partner name="Ucare_openDispenser"/>
</partners>

<variables>
 <variable name="dispenserID"  />
 <variable name="timeToOpen" />
</variables>
<pointcut name="openDispenser" contextCollection="true"> Reminder</pointcut>n
<advice type="before">
             <condition type="localvariable">
                    <variable1 ucare_variable_name="crId"></variable1>
                    <operation ucare_operation= "equal"></operation>
                    <variable2 ucare_variable2_name= "s1015923"></variable2>
             </condition>
             <bpws:sequence>
               <bpws:assign>
                    <bpws:copy>
                      <bpws:from expression="dp21244" type="string" />
                      <bpws:to variable="dispenserID" />
                    </bpws:copy>
                    <bpws:copy>
                      <bpws:from expression="10 minutes" type="string"/>
                      <bpws:to variable="timeToOpen" />
                    </bpws:copy>
               </bpws:assign>

               <bpws:invoke name="invoke" partner="Ucare_openDispenser"
ucare_type="process" />
             </bpws:sequence>
</advice>
</aspect>
```

Table 7 : An aspect file

This aspect file contains the following information:

-*Name of the advice*: this is the process/services written in BPEL that will be invoked if aspect conditions are satisfied.

-*Type of the advice*: there are three types of advices, namely, *before, after and around* which indicate that the advice is executed before, after or instead a joint point activity, respectively[31]. In this example, a *before* type is applied.

-*Variables for the advice*: those are the inputs for the advice. If an advice is considered a function, variables for the advices are the parameters of the function. The values of variables can be read from the business process or can be specified directly in the aspect file.

-*Pointcut*: as mentioned briefly in 5.2.2, pointcut in aspect files is a list of joint points. A joint point is pre-defined point in the process execution[31]. In this example, *Reminder* is a joint point of pointcut *Opendispenser*.

-*Conditional statement*: the conditional statement of aspect files determines whether the advice will be execute or not. In this example, the conditional statement expresses that "if the care-receiver's ID is s1015923, the advice will be executed". This statement is optional in aspect file.

The aspect manager (JavaScript code) read the aspect files and uses the *XML parser* to retrieve the preceding information. In the second step, before executing advices, the aspect manager will check conditions.

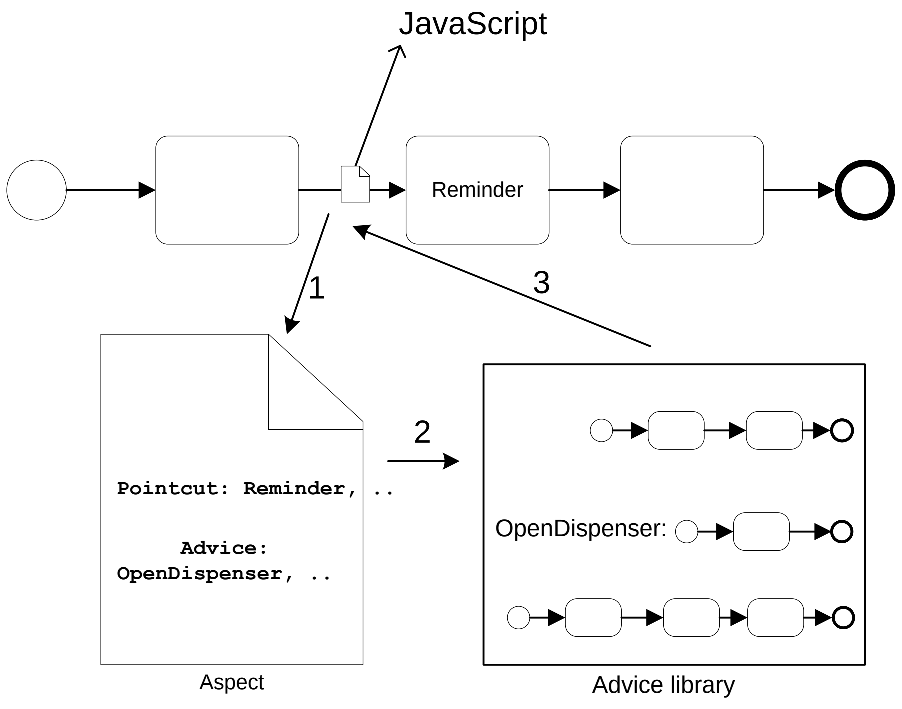
Evaluating aspect conditions

There are two types of condition needed to be checked. First, concerning joint points, the name of the next task will be executed will be compared with the name of *joint points* in *pointcut*. If they are equal, the second check will be performed. For example, if the name of the next task in service plan is Reminder, and there is a joint point, titled Reminder, the first check is passed. The second check is specified in conditional statement mentioned above. If this check is passed, the advice will be executed.

Executing advice

Lombardi allows executing advices that can be business process and business services. At this step, the name of advice and the variables as its input will be used so that the advice can be invoked.

The operation of the aspect manager can be depicted in the following figures.



1. Retrieving aspect information 2. Checking conditions 3. Executing the advice

Figure 11 : Aspect manager

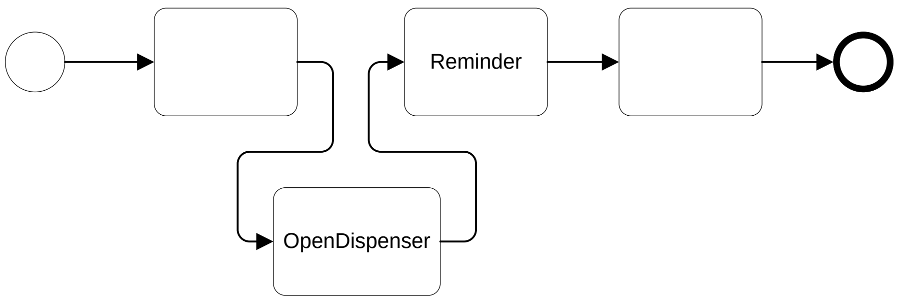


Figure 12 : Result- a service plan

6.3.3 Context database

Context database, as suggested by name, stored the user context information. IBM DB2, a relational model database server, will be used.

6.3.4 Adaptors

Adaptors take the responsibilities of device facilitators and device monitors. In details, exposed as web services, they receive invocations from the process executor and then translate that invocation to the format of the targeted device. In another direction, the device send device information to the aspect manager, e.g., a light in kitchen is broken; the adaptor of that device will convert device-specific data format to the standard format consumed by the system. However, due to the time being, in this prototype, we assume that all adaptors are already available.

6.3.5 Service discoverer

When there are many service providers, the *service discoverer* supports the process composer (care-givers or IT technicians in hospitals) in composing business processes by giving the most suitable services according to the service selection rules[24]. However, with intention to concentrating on dealing with D&D requirements by using the dynamic workflow composition method, we assume that this module is already existed and the composers have and know the most suitable services to use.

6.3.6 Process executor

Needless to say, the *process executor* is the core of the process engine of Lombardi. Process executor - Process Center Server in Lombardi- is depicted as follows.

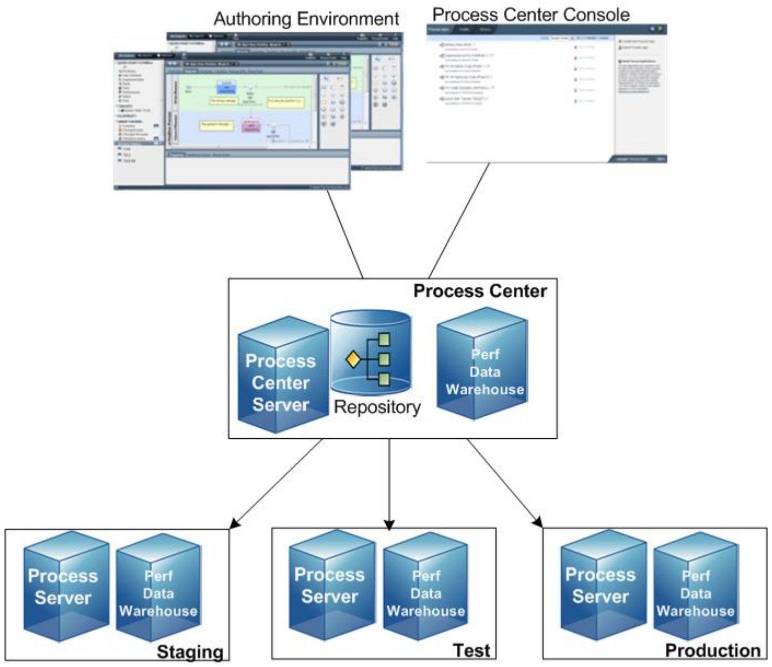


Figure 13 : Lombardi's components [44]

Not only does Process Center allow the composer run their process, but it also stores performance data for the testing purpose.

6.4 Implementation

In the previous part, the way to build each module of the architecture prototype is presented. In this part, in dealing with different sources of dynamicity and diversity, the co-ordination between those modules will be revealed.

6.4.1 The business process

The main business process contains static tasks that will be executed without being influenced by business rules. For the reminder scenario, based on the list of services in section 6.2.3, we have the following business process:

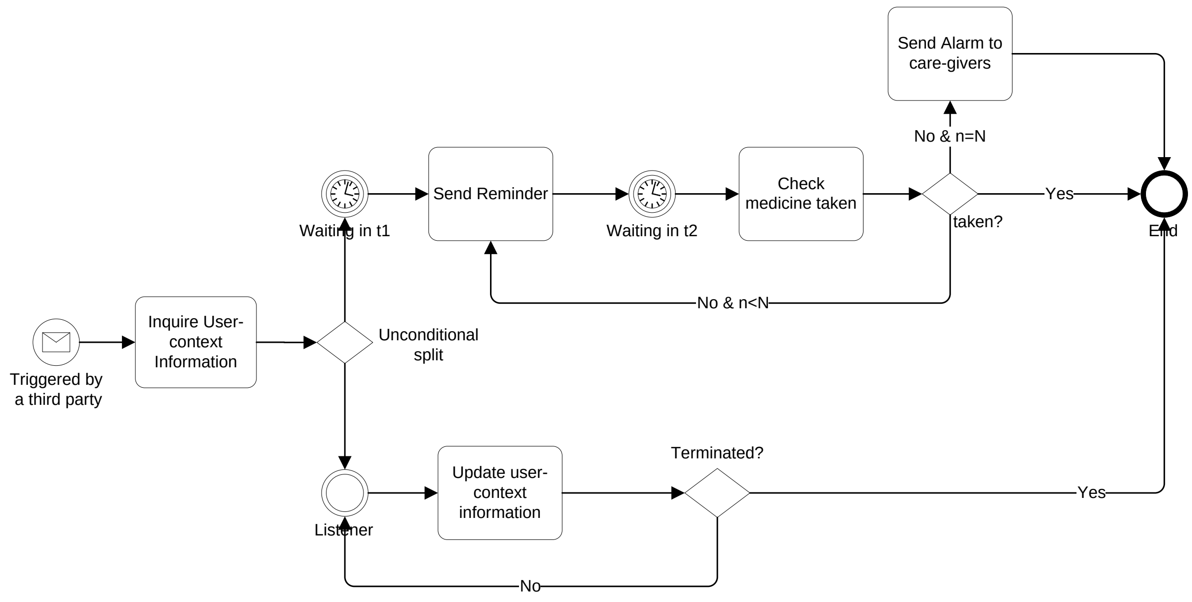


Figure 14 : Reminder Process

Basically, for each care-receiver, the reminder process is triggered by an application of the third parties, e.g., treatment management system of a hospital. After the first inquisition of user-context information, the aspect manager check business rules to initialize the local variables of the process. For example, if the care-receiver is in kitchen, the alarm will be set to “light”. In the same way, based on calculation rules, the aspect manager calculates t1, t2 and endpoint of the web service in each task of the process. T1 is the waiting time from the process is triggered until the first reminder is send. After that, the system waits in t2 before checking whether the medicine is taken or not. If not, the process goes back to reminder task. This loop is executed until the number of sent reminder is equal to a predefined number. In this case, one alarm is sent to care-givers.

In another parallel branch, the context server hears messages from adaptors about changes in the user-context. When a change happens, the aspect manger does the same task as above to calculate new values of parameters in the main process. Then, if the main process still runs, a loop returns to the listening task of context server; the listening task is invoked again.

6.4.2 The behaviors of the system in dealing with user-context dynamicity

The reminder process is ignited by a third party who used it as one part of an application, e.g., treatment software. Implemented by soapUI, the trigger sends a data structure including the identity of the care-receiver, type of event, time and date. After being started up, the behavior of the system, at a specific moment, depends on the updated user-context information. A behavior can be a static task of the reminder process or an advice which is determined at run-time by business rules.

To show how dynamically the system reacts to user-context changes, two situations will be presented. Each situation represents a source of D&D requirements. The first one shows the logic to alter the alarm according to the location changes of a user (context-user dynamicity). The second one presents a specific preference for a user with ID "s101353". This preference show how the system handles different needs of users (diversity of preferences/needs). Since the behavior of systems toward dynamicity of preference and dynamicity of user context are quite similar, we do not present an example of dynamicity of preferences. As mentioned in section 3.1, the user-context information is described as a complex data structure including the user's ID, location, activity and time [19].

Situation 1: dealing with dynamicity of user context

In this situation, the care-receiver does not perform any activities that can cause changes for the reminder process. The context dynamicity is restricted to changes of location. To handling this change, the aspect manager will check business rules to match the location to a corresponding endpoint as follows:

Location		**Endpoint**	**Device**
Kitchen	→	http://**130.89.227.130**:9090/ws/Ucare_WS_notifyReminder/	Lights
Bedroom	→	http://**130.89.227.132**:9090/ws/Ucare_WS_notifyReminder/	Television
Corridor	→	http://**130.89.227.133**:9090/ws/Ucare_WS_notifyReminder/	Telephone

Table 8 Location-endpoint matching services

The cooperation of modules can be depicted in the sequence diagram (see figure 16). The final service plan with all tasks that needs to be done is generated as follows:

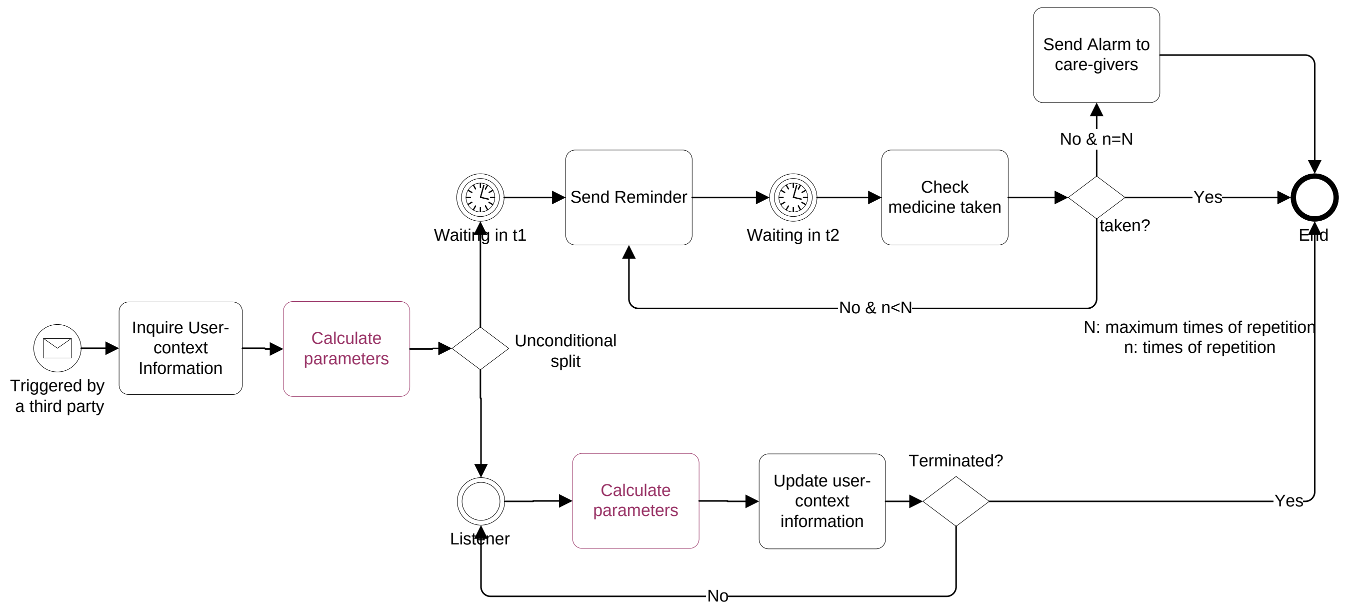


Figure 15: The service plan when dealing with changes in user-context

Handling changes by location-endpoint mapping, however, is merely applicable to support the changes that do not require altering the order of steps or inserting steps in the business process. Two steps, called “Calculate parameters”, are executed by the aspect manager.

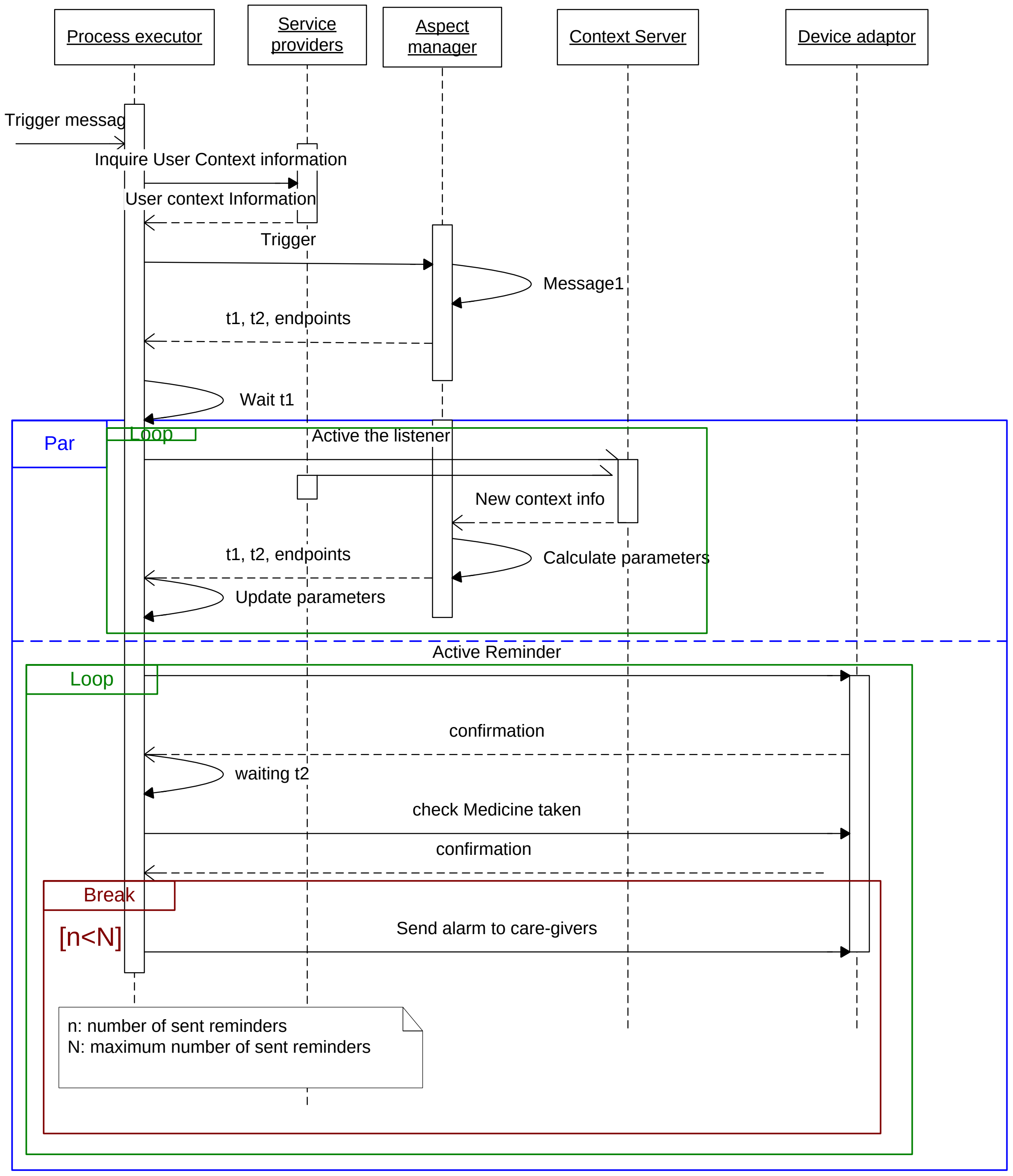


Figure 16 : The sequence diagram when dealing with changes in user-context

Situation 2: dealing with a user's preference

This situation involves the business rule -R1- which reflects the diversity of users' preference. When this business rule is applied, the system reacts by adding a service into the main process. The way to implement it is already explained in the aspect manager.

We suppose that the ID of the care-receiver in the business process is "***s1015923***". His/her preference is formulated by the business rule R1. Then, when R1 is applied, the final service plan generated by the business process and aspect manager can be depicted as follows:

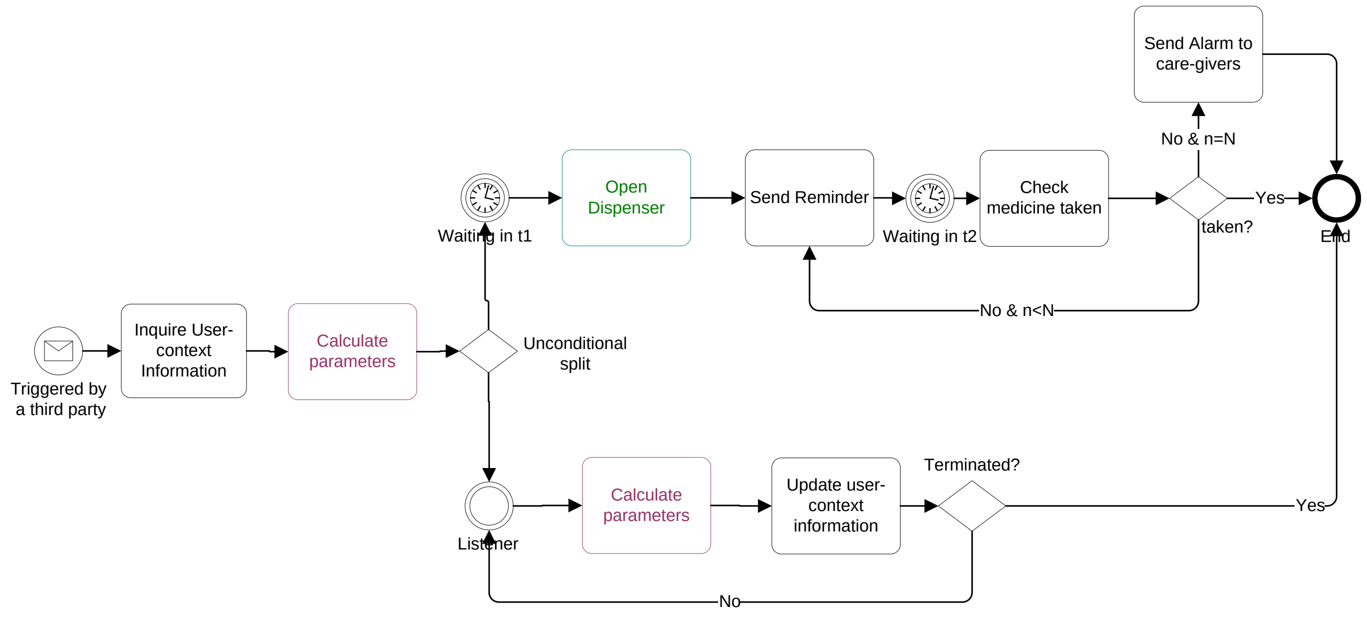


Figure 17 : The service plan when dealing with a user's preference

In this plan, it can be seen that one more task is added due to the preference of the care-receiver.

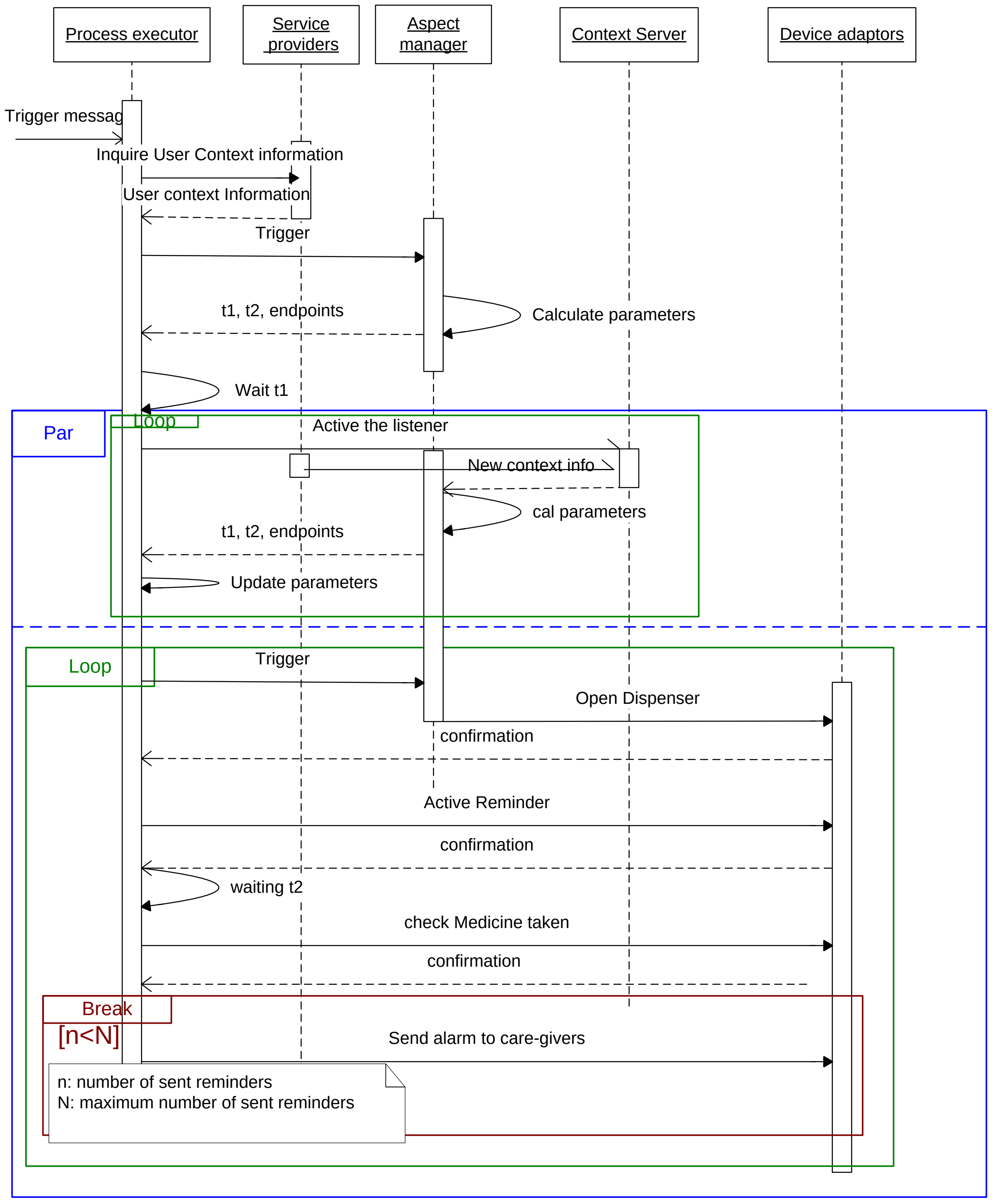


Figure 18 : The sequence diagram when dealing with a user's preference

6.5 Summary

In this chapter, we explore the following content:

- We introduce a scenario to illustrate the capability of the proposed architecture.

- We determine the associated services with the scenario

- After introducing the tools, we present the way to build each modules of the architecture by those tools.

- The modules of the architecture are operationalized in the reminder scenario. By the scenario, we also see how the system works and the coordination between different modules.

Through this chapter, we can conclude about the feasibility of the approach in two senses. First, the approach has ability to dynamically insert the services/processes in the main business process. Second, the approach can be implemented by the current tools.

Chapter 7 Evaluation

This chapter assesses the proposed architecture (chapter 5) that is already implemented in chapter 6 with regard to the ability for supporting D&D requirements. As mentioned in 3.2 about the relationship between user-context awareness and D&D requirements, we argue that D&D requirements can be handled by user-context awareness that allows a system to adapt its behaviors according to changes. In 3.4.2 about the relationship between SOA-based architecture and user-context awareness, we state that a SOA-based architecture, in which a web-service composition method is exploited, can empower user-context awareness by enabling the re-composition of web-services thus adapting the system's behaviors. This line of reasoning refers that the main goal of the purposed architecture is to support dynamicity and diversity. Therefore, evaluating the architecture proposed in chapter 5 is refined to evaluating the re-composing web-services process which supports diversity and dynamicity.

Regarding the ability of adapting to changes in the literature, there is a very close concept: **business process flexibility**. Business process flexibility is defined as "the capability of business process to change"[45]. Therefore, changes in environment like new laws and changes in business strategy can be managed by the process flexibility [46]. In other words, business process flexibility and context awareness have the same purpose of supporting changes. Because of the similarity in the purpose and the abundance of research toward process flexibility evaluation, we apply the methods for evaluating business process flexibility to evaluate the web-service composition which enables context awareness in our architecture.

There are many approaches to measure the flexibility of a business process system. Kasi and Tang [47] propose that process flexibility can be measured in three dimensions: **time**, **cost** and **ease**. This means that, when a change happens, a process is flexible when it takes less time, less cost and maximum ease to adjust. Process flexibility can be also evaluated by investigating the extent of change supported. Following this way, Asuncion, et al [48] employ the taxonomy of criteria proposed by Regev, et al [49] to position their approach of separation of business rules.

To evaluate the purposed SOA-based architecture, the two previous approaches are used. The structure of this chapter is organized in two sections. First, to show the ease of enclosing a new rule, the '**ease**' dimension will be presented. Time and Cost dimensions are important; however, due to the time being, those two dimensions are saved to the future work. Second, we will assess the architecture in terms of its position in the taxonomy of flexibility to see which type and which extent of change supported by the architecture.

7.1 Easiness for adapting the business process

The ease dimension is not easy to be quantified. The Technology Acceptance Model 3 (TAM3) [50] shows causal relationships between 6 factors (computer self-efficacy, perception of external control, computer anxiety, computer playfulness, perceived enjoyment and objective usability) and Perceived Ease of Use. Some of those relationships are moderated by Experience. Consequently, a full measurement of the ease dimension is not realistic due to the time being of this thesis. Therefore, we choose influential factors that are observable in our prototype. Our attention is focused on Objective Usability and Perception of External Control. The first one, Objective Usability, is defined as "comparison of systems based on the actual level (rather than perceptions) of effort required to completing specific tasks". The second one, Perception of External Control, refers to organizational and technical resources for supporting the use of system. Those two factors suggest us to assess the ease in an indirect manner by evaluating how the system supports the care-givers in case of changes. In other words, if the support from the system is good, the level of effort (Objective Usability) is lessened and the technical supporting resources are well designed, leading to increasing ease.

Three sources of D&D requirements, namely, preference/need diversity, context dynamicity, and preference/need dynamicity can be examined from a different perspective. According to K. Kumar [45], there are four types of stimulus for business process flexibility, namely, constant, uncertain but crisply predefined, ambiguous and surprise. With regard to constant stimulus, there are no variations. To the uncertain but predefined stimulus, a finite set of predefined contingencies are determined. Ambiguity of stimulus refers to the difficulties in indentifying the stimulus. A surprise stimulus is not anticipated in the design time. We can map those two classifications as the following table. The constant stimulus is excluded because its response is the unchangeable main business process. For example, in the reminder scenario, the constant stimulus is the triggering message from third parties to start up the reminder process.

	Predefined	Ambiguous	Surprise
Source 1: Preference/need diversity	✓	X	X
Source 2: Preference/need dynamicity	✓	✓	X
Source 3: User-context dynamicity	✓	✓	✓

Table 9: Mapping two classifications

This table expresses that predefined changes (stimulus) can be found at all sources of D&D requirements while ambiguous changes appear in the second and third sources and surprise changes are ignited by user-context changes. We assume that the care-givers are knowledgeable enough to specify users' preferences/needs to avoid ambiguous and surprise changes for this kind of source. However, with difficulties when interpreting complex health conditions (which cause preference/need dynamicity), ambiguous stimulus can emerge. User-context changes occur frequently with contingencies can result in a surprise stimulus.

To demonstrate the extent that the architecture supports the care-givers to update/add business rules when changes appear, we introduce three variations for the Reminder scenario. Concretely, the first variation reflects one predefined changes of the care-receivers. The second variation represents the case in which the system does not know the response due to the ambiguity of health information; nevertheless, by supporting from care-givers, a suitable response is specified.

The last one involves one surprise change that is unsolvable to the system and also strange to care-givers.

7.1.1 Variation 1- Predefined changes

As shown in table 8, predefined changes can be found in three sources of dynamicity and diversity. As the behaviors of the systems towards this kind of change are similar, we give one representative which relates to the diversity of users' preferences/needs.

A predefined preference/need and the system's response are modeled by a business rule in the **design time**. Actually, this kind of variation is presented in R1 in the previous chapter. In details, a preference of a specific care-receiver is transformed into a business rule *"if the care-receiver' ID is "s101hyk", the medicine dispenser will be opened automatically before sending reminder to the care-receiver"*. The reaction of the system including reading aspect files, assessing the conditions and executing the additional services/processes is also mentioned in 6.4.2 about the mechanisms of aspect manager and 6.5 about the coordination of the whole architecture. In this part, we merely focus on the necessary steps in order to introduce this business rule (user's preference) into the system.

Regardless of the details of implementation tools, the procedure for composing a business rule in **design time** consists of three main steps:

Step 1: Transforming the user's preference into one business rule

This step will be conducted with the assistance of care-givers. The care-giver who understands the care-receiver' preference are able to concretize that preference in to a business rule. For example, in this case, in the design phase, the care-giver knows that the care-receiver with ID "s101hyk" cannot open the dispenser by himself. Therefore, one business rule can be formulated as R1: *"if the care-receiver' ID is "s101hyk", the medicine dispenser will be opened automatically before sending reminder to the care-receiver"*.

Step 2: Writing the business rule in one aspect file

Because of the lack of tools to generate aspect files (XML format), the participation of technical staff, e.g., programmers in writing the aspect files is a must. However, this step can be facilitated and accelerated by using templates. For example, the template for the business rule type 1-action enabler- can be described as follows:

```
<aspect name="Name_Of_Aspect"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

<partners>
 <partner name="Name_Of_Advice "/>
</partners>

<variables>
 <variable name="Name_of_variable1"  />
 <variable name="Name_of_variable2" />
```

```
</variables>
<pointcut name="Name_Of_Pointcut" contextCollection="true">
Name_of_JointPoint</pointcut>
<advice type="Type_Of_Advice">
       <condition type="localvariable">
                     <variable1 care_variable_name=”Name_of_Localvariable"></variable1>
                     <operation ucare_operation= "Name_of_operation"></operation>
                     <variable2 ucare_variable2_name= "Value_to_compare"></variable2>
       </condition>
       <bpws:sequence>
                <bpws:assign>
                     <bpws:copy>
                       <bpws:from expression="Value" type="string" />
                       <bpws:to variable="Name_of_Variable1" />
                     </bpws:copy>
                     <bpws:copy>
                       <bpws:from expression="value" type="string"/>
                       <bpws:to variable="Name_of_Variable2" />
                     </bpws:copy>
                </bpws:assign>

         <bpws:invoke name="invoke" partner="Name_of_Advice" ucare_type="process" />
              </bpws:sequence>
</advice>
</aspect>
```

Table 10: A template for aspect files

As can be seen in this template, the way to integrating the business rules in the main process is quite convenient. Because of the separation between rules and processes, the composer can easily specify *when* (the rule’s condition) and *where* (the list of joint points) the rules are applied without being distracted by the possibly complexity of business process.

Step 3: Composing the advice

The advices can be considered as sub-processes that will be invoked by the main process when the business rule’s conditions are satisfied. In other words, the advice of one aspect refers to *what* to do. Normally, advices will be textually written in BPEL. However, many applications permit composting advices in intuitional way. For example, the next figure illustrates the authoring environment of Lombardi.

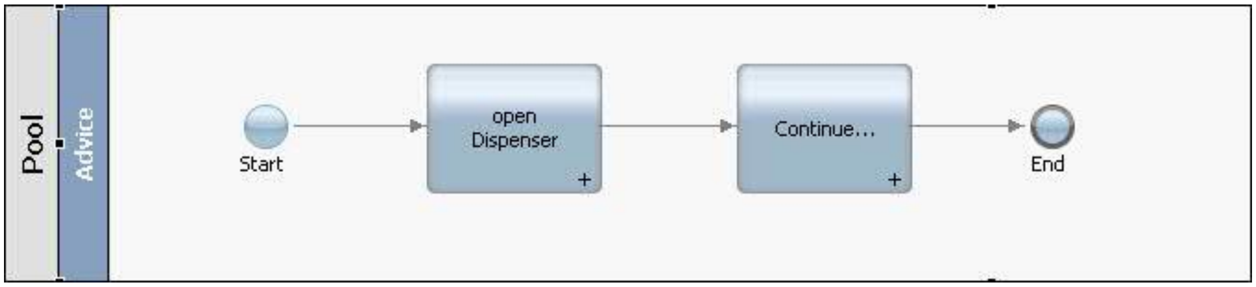


Figure 19: Composing an advice

In our proposed architecture, we suggest that all possible advices should be pre-composed and placed in the advice repository. This way increases the reusability of advice thus saving time to introduce a business rule into the system.

In runtime, when the system encounters a user-context change whose reactions are already defined, the care-givers do not have to do anything.

7.1.2 Variation 2- Ambiguous heath condition information

In the first variation, in dealing with the user's changes, the system knows to react because those changes are determined in the design time. The second variation, the way to handle a change is already existed; however, the system has difficulties in interpreting the change to match it into one rule to fire an advice. In this case, the care-givers can help by clarifying the ambiguous information so that the system can make a choice. The following example shows what care-givers have to do to give instructions.

> *"Daily, based on the hearing condition, the reasoning service decides the volume of sound devices. However, one day, the care-receiver forgets to update this information. The reminder process is activated."*

This is the normal possible event, the care-givers know how to dealing with this and already prepared the solution in the form of an advice. However, the care-givers, accidentally, forgot to make clear the solution to this situation by business rules. The out-of-date hearing information confuses the aspect manager.

In this case, the system recognizes that the event is in hearing problem which is not urgent issue. For the care-givers, as the advice is already composed, the only step should be completed is *adding a business rule*. The details of that step are described in the step 2 (writing business rules) of the variation 1.

The maintaining task in which the solutions and rules are known also can be classified in this group.

7.1.3 Variation 3- Surprise changes

In the first and second variation, the change is predefined, and the corresponding reactions are specified in aspect files in design time. This variation represents the third type of change/stimulus – changes that are unknown by the system and unforeseen by care-givers.

Due to the safety criteria, the dynamic-workflow composition method cannot generate service plans that are out of control of the care-givers. As a consequence, when facing with an undefined change, the system is not able to decide its response by itself. This requires interaction with care-givers who have sufficient domain knowledge to guide the system. The desired advantage of the architecture is supporting the care-givers in introducing the response faster. To illustrate that ability of the architecture, in the Reminder scenario, we create the imaginary event that is unfamiliar to the system.

> *"The care-receiver with ID m106ksg has to take his medicine at 10.50 Pm. One day, his has a birthday party at 10.30 Pm at his friend's house in another city. He decide to take the medicine at 10.00 Pm so that his has time to move"*

We suppose that by sensors inside the dispenser, the 3rd party is aware about that. The Context Server receives a message informing the event. As the response for this event is undetermined and there is no related information about this, a default plan is triggered. Some default plan, so-called migration strategies, can be taken into account like *forward recovery, backward recovery, proceed, transfer* [46]. In this case, due to the critical safety requirements, the *backward recovery* strategy is most suitable because the affected process instances are canceled, and restarted. Furthermore, an emergency action plan needs to be performed to assist the system. For example, the plan will save the current user-context information into the database for auditing purposes, and send the emergency signals to the care-givers for asking helps. This emergency action plan can be modeled as an advice invoked by aspect manger.

In this section, we focus solely on how the system supports the care-givers in introducing new changes. Therefore, we assume that:

- The affected process is aborted;
- The emergency action plan is triggered;
- One care-giver calls the care-receiver to suggest him to bring the medicine and take it strictly at 10.50;
- One care-giver is about to create new business rule to dealing with this kind of event.

The task of the care-giver in this variation is similar to the one in the first variation. They need to fulfill the three main steps, namely, transforming the user's preference into business rules, writing the aspect file and preparing the advice.

It is worth mentioning that the critical level of the change is important to decide the migration strategy, and not all changes are crucial. For a less critical but still strange user-context change, for example, the care-receiver cannot open the door, a *proceed* strategy which ignores the change[46], is more suitable. However, the problem of deciding which strategy should proceed is not in the scope of this thesis and not supported by the architecture. The architecture just helps to facilitate updating new business rules.

In conclusion, the system's supports which influence Ease of use can be summarized in three points. First, due to the separation of business processes and business rules, the architecture allows to compose advices separately in GUI design tools, alleviating the burden in creating new rules (in the third step). Similarly, this feature allows reusing advices stored in a repository. Second, because of the aspect approach which overcomes the crosscutting concern, maintaining business rules in case of change is easy. Lastly, the clear structure of aspect XML files makes it possible to be generalized in a set of templates, relieving effort in the second step. The following table summarizes the system's support.

Type of changes	**Supports from the system**	**Assistance of care-givers**	**Assistance of technicians**
Predefined changes	No further efforts needed to be done.	No	No
Ambiguously interpreted changes	Simplifying the connection between advices and problems by aspect views. Enabling reusability of templates of aspect files	Yes	Yes
Surprise change	Simplifying the connection between advices and problems by aspect views. Enabling reusability of advices and aspect files.	Yes	Yes

Table 11: summarizing system's supports

7.2 Positioning the supported flexibility

Based on the taxonomy of business flexibility introduced by Regev, et al. [49], we position the our architecture. By this way, the abstraction level of change, subject of change and properties of change are determined. From this perspective, positioning can be considered as part of evaluation because it shows us which extent of flexibility and sources of changes the architecture supports. Three dimensions of the taxonomy will be discussed in this section.

7.2.1 Abstraction level of change

The abstraction level of change refers to changes of business process and changes of process instances[49]. According to this differentiation, in our proposed architecture, which business processes is fixed and business rules are tailored according to preferences of care-receivers, the changes at the process instance is supported.

7.2.2 Subjects of change

The functional perspective is not supported because the purpose of a business process does not change. By dynamically changing endpoints and adding/moving business services in aspect files, the architecture allows altering activities, which are not a part of fixed business processes. Therefore, *the operational perspective* is supported. *The behavioral perspective* specifies the conditions and the moment to perform activities[49]. Those conditions and moment are modeled as conditions and joint points in the aspect files, supporting the behavioral perspective. The architecture does not have dedicated module to handle the incompatibility of messages transferring between activities, leading to discourage informational perspectives. The participants keep their roles during the execution of a process; therefore, the organizational perspective is not a must and not supported by the architecture.

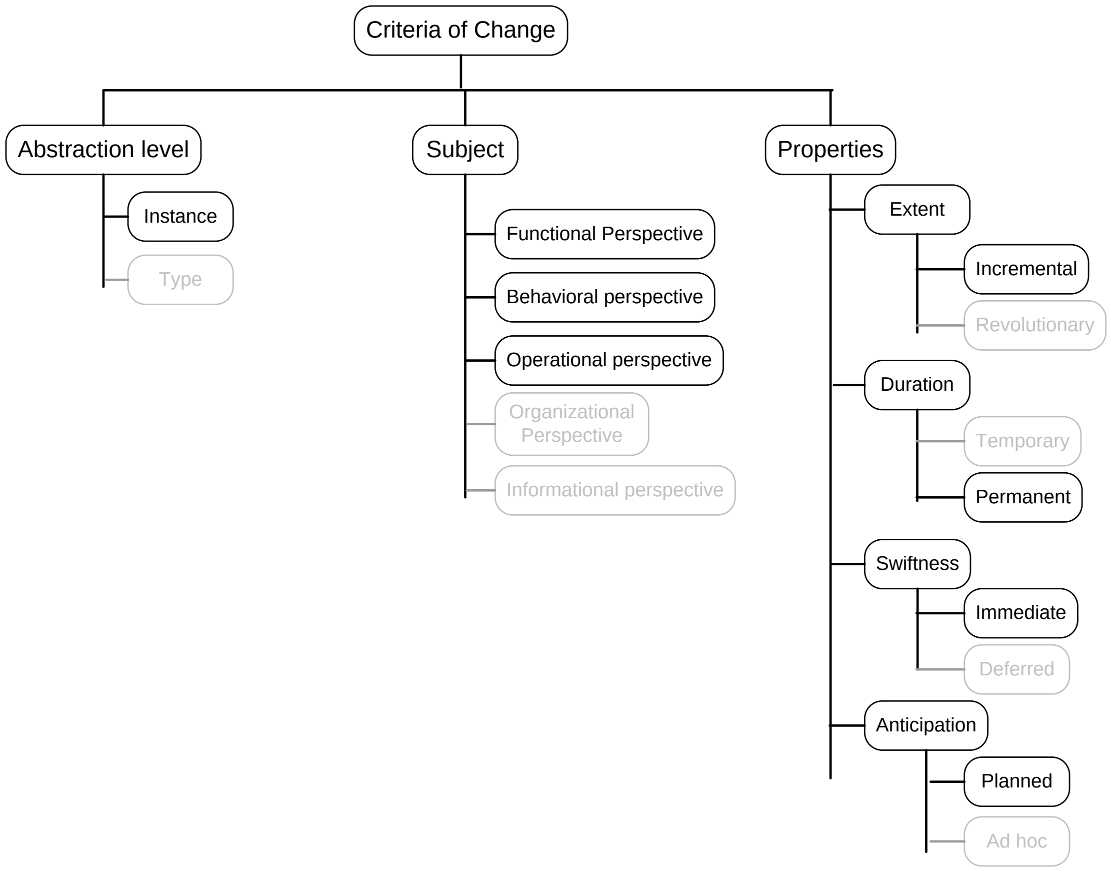


Figure 20: Positioning the architecture's ability

7.2.3 Properties of change

There are four properties of change: the extent, the duration, the swiftness and the anticipation of change. It can be seen that, for the extent of change, only *incremental changes* are supported because the main purpose of each process is consistent and the care-givers can only modify the dynamic part. Concerning *the duration of change*, after applying one rule, that rule is permanently kept in the rule repository to be called in the next time, supporting the permanent change. About *swiftness of change*, when a business rule is created, it is activated for all running instances of the process. Therefore, the immediate type of change is supported. Changes which are pre-defined in the design time is fully controllable (variation 1 and 2) while exceptional changes require many efforts of care-givers to compose then update new plan for dealing with them. In other words, the architecture can handle planned changes.

7.3 Summary

Through the evaluation this chapter, we can have the following conclusion about the proposed architecture:

-The architecture supports well the changes that are pre-defined. In the procedure to include a business rule that represents a change, technicians are necessary to convert the business rule into an aspect files and compose the advice (at design time). However, as we mentioned, the easiness of this procedure can be improved much by create a library of advices and templates to reuse.

-For the confusing changes (those needs the interpretation of care-givers) the care-givers just need to edit the aspect file.

-For the exception, an emergency action plan, modeled as an advice, will be called. The care-givers have to conduct three steps. This kind of change take time, effort and also difficult to handle.

-The ability of the architecture for handling D&D requirements is positioned (figure 20).

Chapter 8 Discussion, Recommendation and Conclusion

This final chapter ends the report by giving the discussion, the conclusion and some recommendations for the future research. In the discussion section, based on the evaluation in chapter 7, we figure out the advantages and the weaknesses of the proposed architecture. Then, some recommendations will be given to improve the quality of the architecture, making it more persuadable to bring the architecture into reality. Finally, the conclusion closes the report.

8.1 Discussion

As mentioned above, base on the evaluation of the architecture, in this section, we will discuss about its advantages and weaknesses.

8.1.1 Advantages

The proposed architecture is the combination of two main ideas: user-context awareness and the web service composition method. It, therefore, inherits their strength to deal with D&D requirements.

First, it is worth mentioning that the interoperability is strongly supported by Service Oriented Architecture (appendix B). Therefore, it is obvious that the interoperability-the technical requirement for Ucare- is likely backed by our architecture which is able to connect different services with different specification, technologies from various providers by adapters and Enterprise Service Bus. However, as also pointed out in [18], the semantic interoperability is sill problem and need further research.

Second, with regard to with dynamicity and diversity, our solution offers the following features:

- It allows separating business rules and business process, making it possible to reuse business rules, advices (stored in library) to different individuals. Therefore, in case of undefined changes, the care-givers can save time by reuse rules and advices.

- Regarding the maintenance, it is easy for the caregiver to update business rules and business process. Just based on the logic of joint points and advices, they do not need to have deep understanding about possibly complex business processes. This feature is helpful a lot with frequent changes in business rules.

- Regarding the flexibility, although the process, the advices are composed at the design time; the advices are inserted in the process at run time. The business rules specify the conditions to insert, and they (business rules) can be modified at run time. Therefore, the whole composition process is still flexible to some extent and there are no needs to restart the produced processes when they are started.

- Regarding the safety criteria, the architectures assure that whatever the system does to handle a change reflects exactly what the care-givers want. There are no chances for any unexpected reactions. The safety is well supported.

- Regarding the ability to gather context information, with a listening task deploying in a parallel process, the architecture is able to get the most up-to-dated context information from third parties.

- In dealing with different type of changes, as evaluated in the evaluation, the web service composition method support well predefined changes and confused changes. The three steps needed to be performed for group 1, obviously, make the impression of difficulties. This is true to compose a totally new rule to dealing with change of care-receivers. However, a valuable remark is that a totally new change that requires those three steps is the worst case. The very frequent changes happen with business rules that only require updating business rules in the aspect files. This updating process, in other words maintenance, is supported by splitting business rules and business processes as discussed above.

8.1.2 Limitation

However, the current problems of dynamic workflow web composition method are still existed in our architecture, causing the following issues.

- About non-intrusiveness criteria, looking at the behaviors of the system in dealing with three variations (chapter 7), we can observe that there are no interaction needed to be made between the care-receivers and the system. However, for the care-givers who manage the business rules, business processes, the repetitiveness of intervention of care-giver is high. This can be considered the downside of safety- when care-givers want to control every activity, they are surely got involved.

- The dynamic web service composition does not support the exceptional changes (variation 3, chapter 7). Therefore, in case of an exception, there are many works to do like firing the emergency plan, and composing new business rule according to the exception.

- Although the externalization of business rules and business processes, the introduction of library of advices and pre-build templates can accelerate and facilitate the maintenance and the creation of new advices and rules, many efforts and long time to respond to one change are unavoidable. Furthermore, due to the lack of supporting tools to work with aspect files, the assistances of technicians are vital (but not always available).

- Relating to interoperability, our architecture misses a module (adapter) to standardize the services from different service providers. However, as mentioned in the scope, this does not relate to our focus.

- As pointed out Charif et al [31], when there are a number of business rules, checking rule consistency, combining rules, solving conflict are difficult to manage manually. Therefore, a business rule engine is necessary. Furthermore, coming back to four type of business rules (5.2.1), it can be seen that the inference type of business rules are hard to be modeled by aspects [31].

8.2 Recommendations

From the previous discussion, it appears that there are many rooms to improve the architecture. The following recommendations target to all related aspects of the architecture:

First, to save time and effort in creating/updating an aspect file, especially for non-technical staff, instead of working with XML files, a Graphical User Interface (GUI) module is very useful in helping to create new aspect files, update the existing aspect etc.

Second, we suggested that a library of advices and library of XML template for creating new aspects file can save much time and efforts. Therefore, to exploit the power of libraries effectively, the classification of advices and templates is essential. For example, it would be very easy to users if all advices needed to handle hearing problems are grouped into one set. This could help the users can find the advices faster. However, this seems to be difficult task, e.g., due to the complexity of the domain, one advice can be applied to many problems. A further research about this topic is necessary.

A business rules engine solution is rejected from our architecture due to its complexity to non-technical staff, its costly price and its limitations in composing and executing business processes. However, there are some advanced functionalities of business rule engines. For example, it is the ability of expressing the business rules in the nature language. This could be helpful to build one software/service that can do that task, and then integrate to the system.

8.3 Conclusion

These few lines of the conclusion will summarize the whole report and also give some ideas about the contributions of the thesis. Staring with a subset of problems of the Ucare project, we plan this research carefully by a list of research questions. After that, the reminder of the report is to find the answers, connect the answers and explore new knowledge. The final and most important results are a SOA-based architecture design –result of marriage of context awareness and service composition- and the evaluation of this architecture in terms of its flexibility.

To practical implication, the thesis provides a good architecture which stems from two reference context-awareness architectures and novel ideas of SOA. Furthermore, the feasibility of this architecture is proved in the implementation part (chapter 6); its flexibility is assessed in the evaluation part (chapter 7). In addition, the implementer of this architecture is of benefit from the discussion part in which the weaknesses, along with the corresponding suggestions, are addressed.

Regards theoretical contributions, although there are some limitations, the prototype of our architecture demonstrate the great applicability of service oriented architecture in to the home-care domain.

1. Health-EU. *Elderly*. 2011 08/03/2011; Available from: http://ec.europa.eu/health-eu/my_health/elderly/index_en.htm.
2. Norbert, M., O. Rukiye, and C.G. MARCELINO, *Active Ageing and Independent Living Services, The Role of Information and Communiation Technology*. 2010, Institute for Prospective Technological Studies.
3. Commission, E. *Information Can Save Your Life*. 2007 03/2007 17/3/2011]; Available from: http://ec.europa.eu/information_society/tl/qualif/health/index_en.htm.
4. Commission, E., *e-Health - making healthcare better for European citizens: an action plan for a European e-Health Area* 2004.
5. Commission, E. *The right prescription for Europe's eHealth*. 2010 25/10/10 [cited 17/3/2011; Available from: http://ec.europa.eu/information_society/activities/health/policy/index_en.htm.
6. Ucare-Project. *Ucare Background*. 2008 [cited 2011 16/3]; Available from: http://www.utwente.nl/ewi/ucare/background/.
7. Alireza, Z., et al., *Toward Dynamic Service Provisioning in the Homecare Domain*. 2011, Department of Electrical Engineering Mathematics and Computer Science.
8. Eslami, M.Z. and M.V. Sinderen. *Flexible Home Care Automation*. in *Proceedings of PervasiveHealth 2009 conference*. 2009.
9. McBryan, T., M.R. McGee-Lennon, and P. Gray, *An integrated approach to supporting interaction evolution in home care systems*, in *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*. 2008, ACM: Athens, Greece. p. 1-8.
10. Klooster, J.W.v.t., B.J.F.V. Beijnum, and H.J.Hermens, *U-care: requirements elicitation for ambient assisted living.* IEEE-EMBS Benelux chapter symposium, 2009.
11. Papazoglou, M. and W.-J. van den Heuvel, *Service oriented architectures: approaches, technologies and research issues.* The VLDB Journal, 2007. **16**(3): p. 389-415.
12. Erl, T., *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. 2005, Prentice Hall.
13. Group, T.O.W. *The SOA Work Group : Definition of SOA*. [cited 2011 06 April].
14. Tsai, W.T., et al., *Architecture Classification for SOA-Based Applications*, in *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. 2006, IEEE Computer Society. p. 295-302.
15. OASIS. *Web Services Business Process Execution Language Version 2.0*. [cited 2011 April 09]; Available from: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738479.
16. Schmidt, M.-T., et al., *The enterprise service bus: making service-oriented architecture real.* IBM Syst. J., 2005. **44**(4): p. 781-797.
17. Chappell, D., *Characteristics of an ESB*, in *Enterprise Service Bus: Theory in Practice*. 2004, O'Reilly.
18. O'Brien, L., P. Merson, and L. Bass, *Quality Attributes for Service-Oriented Architectures*, in *Proceedings of the International Workshop on Systems Development in SOA Environments*. 2007, IEEE Computer Society. p. 3.
19. Dey, A.K., G.D. Abowd, and D. Salber, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.* Hum.-Comput. Interact., 2001. **16**(2): p. 97-166.
20. Bricon-Souf, N. and C.R. Newman, *Context awareness in health care: A review.* International Journal of Medical Informatics, 2007. **76**(1): p. 2-12.
21. Gu, T., H.K. Pung, and D.Q. Zhang, *Toward an OSGi-Based Infrastructure for Context-Aware Applications.* IEEE Pervasive Computing, 2004. **3**(4): p. 66-74.
22. Fujii, K. and T. Suda, *Semantics-based context-aware dynamic service composition.* ACM Trans. Auton. Adapt. Syst., 2009. **4**(2): p. 1-31.

23. Rao, J. and X. Su, *A Survey of Automated Web Service Composition Methods*, in *Semantic Web Services and Web Process Composition*, J. Cardoso and A. Sheth, Editors. 2005, Springer Berlin / Heidelberg. p. 43-54.
24. Casati, F., et al., *Adaptive and Dynamic Service Composition in eFlow*, in *Advanced Information Systems Engineering*, B. Wangler and L. Bergman, Editors. 2000, Springer Berlin / Heidelberg. p. 13-31.
25. Schuster, H., et al., *Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes*, in *Advanced Information Systems Engineering*, B. Wangler and L. Bergman, Editors. 2000, Springer Berlin / Heidelberg. p. 247-263.
26. Ghallab, M. and e. al, *PDDL-The Planning Domain Denition Language*. 1998.
27. Sirin, E., et al., *HTN planning for Web Service composition using SHOP2*. Web Semantics: Science, Services and Agents on the World Wide Web, 2004. **1**(4): p. 377-396.
28. E. Sirin, J.H., and B. Parsia., *Semi-automatic composition of Web services using semantic descriptions*, in *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*. 2002.
29. Garde, S. and P. Knaup, *Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology*. Requirements Engineering, 2006. **11**(4): p. 265-278.
30. Lang, A., N. Edwards, and A. Fleiszer, *Safety in home care: a broadened perspective of patient safety*. International Journal for Quality in Health Care, 2008. **20**(2): p. 130-135.
31. Charfi, A. and M. Mezini, *Hybrid web service composition: business processes meet business rules*, in *Proceedings of the 2nd international conference on Service oriented computing*. 2004, ACM: New York, NY, USA. p. 30-38.
32. Hull, R., et al., *E-services: a look behind the curtain*, in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2003, ACM: San Diego, California. p. 1-14.
33. BRG. *Defining business rules- What are they really?* 2000; White paper:[Available from: http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf.
34. Cibrán, M.A. and B. Verheecke, *Dynamic business rules for web service composition.* In R. E. Filman, M. Haupt, and R. Hirschfeld (eds), Proc. of the Second Dynamic Aspects Workshop (DAW05),, 2005: p. 13-18.
35. Halle, B.v., *Business Rules Applied: Building Better Systems using the Business Rules Approach*. 2001: Wiley.
36. Charfi, A. and M. Mezini, *AO4PBEL: an aspect-oriented extention to BPEL.* World Wide Web, 2007. **10**: p. 309-344.
37. van Eijndhoven, T., M.E. Iacob, and M.L. Ponisio. *Achieving Business Process Flexibility with Business Rules*. in *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*. 2008.
38. Rosenberg, F. and S. Dustdar, *Business Rules Integration in BPEL " A Service-Oriented Approach*, in *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*. 2005, IEEE Computer Society. p. 476-479.
39. Beer, T., et al., *Exploiting E-C-A rules for defining and processing context-aware push messages*, in *Proceedings of the 2007 international conference on Advances in rule interchange and applications*. 2007, Springer-Verlag: Orlando, Florida. p. 199-206.
40. Charfi, A., *Aspect-Oriented Workflow Languages: AO4BPEL and Applications*, in *Fachbereich Informatik*. 2007, TU Darmstadt: Darmstadt.
41. Group, T.O. [cited 2011 July 11]; Available from: http://www.archimate.nl/en/about_archimate/.
42. Mendix. 2011 [cited 2011 June 01]; Available from: http://www.mendix.com/product/features/.
43. IBM-InfoCentre. *Lombardi tasks*. 2011 [cited 2011 June 01]; Available from: http://publib.boulder.ibm.com/infocenter/wle/v7r2/index.jsp.
44. IBM-InfoCentre. *Managing the Process Center repository*. [cited 2011 June 04]; Available from: http://publib.boulder.ibm.com/infocenter/wle/v7r2/index.jsp.

45. Kumar, K. and M.M. Narasipuram. *Defining Requirements for Business Process Flexibility*. in *Workshop on Business Process Modeling, Development, and Support (BPMDS'06) Requirements for flexibility and the ways to achieve it (CAiSE'06)*. 2006. Luxembourg.

46. Schonenberg, H., et al. *Towards a Taxonomy of Process Flexibility*. in *CAiSE Forum*. 2008: CEUR-WS.org.

47. Kasi, V. and X. Tang. *DESIGN ATTRIBUTES AND PERFORMANCE OUTCOMES: A FRAMEWORK FOR COMPARING BUSINESS PROCESSES*. in *Proceedings of the 2005 Southern Association of Information Systems Conference*. 2005.

48. Asuncion, C.H., M.-E. Iacob, and M.J. Sinderen van, *Towards a flexible service integration through separation of business rules*, in *14th IEEE International EDOC Enterprise Computing Conference, EDOC 2010*. 2010, IEEE Computer Society: Vitoria, Brazil. p. 184-193.

49. Regev, G., P. Soffer, and R. Schmidt, *Taxonomy of Flexibility in Business Processes*, in *Pro. 7th Workshop on Businss Process Modeling, Development and Support*. 2006.

50. Venkatesh, V. and H. Bala, *Technology Acceptance Model 3 and a Research Agenda on Interventions.* Decision Sciences, 2008. **39**(2): p. 273-315.

51. Davenport, T.H., J.E. Short, and R. Sloan School of Management. Center for Information Systems, *The new industrial engineering : information technology and business process redesign*. 1990, Cambridge, Mass.: Center for Information Systems Research, Sloan School of Management, Massachusetts Institute of Technology.

52. Milanovic, N. and M. Malek, *Current solutions for Web service composition.* Internet Computing, IEEE, 2004. **8**(6): p. 51-59.

Appendix A: List of pilot scenarios

Scenario1- Medicine reminder

Jan is an elderly who lives inside an apartment which equipped with homecare applications on top of our application platform. He should intake his medicine at 11:50 PM. The application should remind him to intake medicine several times up to 15 minutes later than the scheduled time. If not taken, the alarm should be sent to the care center. He has hearing impairment developing over the time. He also uses wheelchair, so the doors inside the apartment should be opened automatically. He only can speak Dutch and prefers to take medicine from the closest medicine dispenser (MD) at night. Two MDs, filled with the required medicine, have been installed, one in the kitchen and the other one in the corridor. The MD inside the kitchen has embedded light. TV inside the sleeping room, all the lights of the apartment and a wristwatch can be used as reminder devices for taking medicine. He prefers not to be reminded by lights after midnight. Nancy, as a care-giver, wants to create a service plan. Because she knows his requirement and situation better than IT specialists, she is going to make a service plan in order to assist Jan and remind him to take his medicine on time. She is going to make the service plan based on Jan's requirements, abilities and preferences.

Scenario 2-John and Marie are patients with a minor form of COPD (Chronic obstructive pulmonary disease). Their quality of life is improved when being active and regulating their weight. However, when being active, for example walking, it is important to preserve their oxygen saturation for safety reasons. If the saturation level drops too low, exacerbation may occur, leading to hospitalization and more expensive long-term care. John has hearing disorders while Marie has vision disorders. Both of them have forgetfulness and need to be reminded for their tasks. Nancy is the professional care giver and responsible to create and tailor the homecare services of the U-Care system installed in their care home. The UCare system employs a Tablet PC, i.e., Tablet, PDA, medicine dispenser and measurement services. MobiHealth, as one of the project partner, encapsulates measurement devices such as oxygen saturation meter and provides measurement services which can be used by other homecare services. Nancy creates two reminder services, (1) to remind them to attach the oxygen saturation meter when they leave their homes to walk outside and (2) to remind them to take their medicine three times per day. Nancy tailor the reminder services to use tablet preferably for Marie because of its large screen and to use higher voice volume either on Tablet or PDA for John due to their vision and hearing disorders, respectively. The reminder services should also take John and Marie preferences into account. For instance, John prefers to get vibration reminder on PDA instead of voice, when his family comes to visit him and he is not alone. If John or Marie does not connect the oxygen saturation meter when they leave the care home or while the device being connected and the saturation drops below a predefined threshold, the U-care system trigger a task, for instance, sending an alarm to the care center. The predefined thresholds and corresponding task are different for John and Marie and must be comply with the existing medical protocols employed in the care center. In case of frequently failed reminders, the U-Care system will inform Nancy to talk to the patients and tailor the services, for instance, because of hearing impairment development for John, it's needed to increase the default volume of reminder voice. Nancy should visit John and Marie every day at their care homes and based on their health condition, she might need to drop a particular medicine intake and thus, its reminder service.

Appendix B: Quality attributes of SOA

SOA Approach's Impact on Quality Attributes [18]

The status column represents the level of maturity SOA in supporting quality attributes. The color red means that SOA solutions are immature and significant effort is required. The color green refers to high maturity of SOA solutions. The color yellow reflects that some SOA solutions support the quality attribute properly but further research is needed.

Quality Attribute	**Summary**	**Status**
Interoperability	Through the use of the underlying standards, an SOA provides good interoperability technology-wise overall, allowing services and applications built in different languages and deployed on different platforms to interact. However, semantic interoperability is not fully addressed. The standards to support semantic interoperability are immature and still being developed.	Green
Reliability	Potentially, problems can occur in many areas, but the use of the underlying standards (WS-Reliability and WS-ReliableMessaging) should mean that messages are transmitted reliably. Service reliability is still an issue as with any element in an architecture.	Yellow
Availability	It is up to the service users to negotiate an SLA that can be used to set an agreed-upon level of availability and to include penalties for noncompliance with the agreement. Also, if a service provider can build into its applications contingencies such as exception handling when an invoked service is not available (dynamically locating another source for the needed service), availability would not decrease and could actually be improved as compared with other architectural approaches.	Yellow
Usability	Usability may decrease if the services within the application support human interactions with the system and there are performance problems with the services. It is up to the services users and providers to build support for usability into their systems.	Yellow

Security	The need for encryption, authentication, and trust within an SOA approach requires detailed attention within the architecture. Many standards are being developed to support security, but most are still immature. If these issues are not dealt with appropriately within the SOA, security could be negatively impacted.	Red
Performance	An SOA approach can have a negative impact on the performance of an application due to network delays, the overhead of looking up services in a directory, and the overhead caused by intermediaries that handle communication. The service user must design and evaluate the architecture carefully, and the service provider must design and evaluate its services carefully to make sure that the necessary performance requirements are met.	Red
Scalability	There are ways to deal with an increase in the number of service users and the increased need to support more requests for services. However, these solutions require detailed analysis by the services providers to make sure that other quality attributes are not negatively impacted.	Yellow
Extensibility	Extending an SOA by adding new services or incorporating additional capabilities into existing services is supported within an SOA. However, the interface/formal contract must be designed carefully to make sure that it can be extended, if necessary, without causing a major impact on the service users.	Green
Adaptability	The use of an SOA approach should have a positive impact on adaptability, as long as the adaptations are managed properly. However, the management of this quality attribute is left up to the service users and providers, and no standards exist to support it. This attribute must be managed in coordination with other quality attributes including stability, performance, and availability, and the necessary tradeoffs must be identified and made.	Yellow
Testability	Testability can be negatively impacted when using an SOA due to the complexity of the testing services that are distributed across a network. Those services might be provided by external organizations where access to the source code is not available, and if they implement runtime discovery of services, it may be impossible to identify which services are used until a system executes. It is up to the service users and providers to test the	Red

	services, and very little support is currently provided for the end-to-end testing of an SOA.	
Auditability	Auditability can be negatively impacted if the right capabilities for end-to-end auditing are not built into the system by the service users and if capabilities are not incorporated into the services by the service providers.	Red
Operability and Deployability	Operating and deploying services and systems that use services need to be managed carefully to avoid problems. The interactions and tradeoffs among this and other quality attributes need to be monitored and managed.	Yellow
Modifiability	Modifiability of services or an application that uses services is directly supported using an SOA approach. However, a service interface must be designed carefully because the changes that impact service users might be difficult to identify if the service is externally available.	Green

List of Figures

List of tables

B

Business rule: a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business.

Business process: a set of logically related task performed to a defined business outcome[51].

Binding port: endpoint

C

Crosscutting concerns: one process addresses several concerns and the implementation of a single concern appears in many places in the process definition[31]

E

Enterprise service bus: open, standards-based message bus designed to enable the implementation, deployment, and management of SOA-based solutions with a focus on assembling, deploying, and managing distributed SOA" [11].

Endpoint: Endpoints in the ESB provide abstraction of physical destination and connection information (like TCP/IP hostname and port number) [11].

S

Service composition: Web service composition lets developers create applications on top of service oriented computing's native description, discovery, and communication capabilities[52].

Service block: a web service

Service plan: is result of the combination of business rules and a business process. Another equal term is service composition.